

POZNAN UNIVERSITY OF TECHNOLOGY

Pattern-based clustering and classification of XML data

Maciej Piernik

A dissertation submitted to
the Council of the Faculty of Computing
in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Supervisor: Prof. Tadeusz Morzy, Ph. D. Dr. Habil.

Poznan, Poland
2015

This dissertation is dedicated to my parents.

Acknowledgements

First of all, I would like to thank my supervisor, Professor Tadeusz Morzy, for his invaluable guidance and tremendous support. Throughout the last five years, Professor Morzy has offered me much more of his time than I could ever ask for. He has been my mentor and has directed me both in terms of my professional as well as personal development. This work, most definitely, would not be possible without his guidance. Thank you, Professor.

I am also very thankful to my research colleague and my dearest friend, Dariusz Brzezinski, who has collaborated on a substantial part of this work. He is one of the best people I know and has always inspired me to be a better researcher and a better person. I would also like to thank Doctor Mateusz Pawlik and Professor Nikolaus Augsten for their valuable insight into the development of partial tree-edit distance measure.

Furthermore, I would like to thank my family, for their constant support and unconditional love throughout my life. I am especially thankful to my parents, for always making sure that I have nothing to worry about and supporting me in my every undertaking. Mother, Father — my every achievement has your names on it. I would also like to thank my younger sister, whose devotion to studying and hunger for life has always been a great inspiration for me. I direct special thanks to my grandfather Tadeusz, for encouraging me to pursue a career in science, helping me on every level of my education, and always believing in me. Last but not least, I want to thank my fiance, for her love, support, and patience. Thank you all.

Finally, I acknowledge the support of the Polish National Science Center under grant no. DEC-2011/01/B/ST6/05169.

Contents

Acknowledgements iii

1 Introduction 1
- 1.1 Applications of XML clustering and classification . . . 3
- 1.2 Open challenges in XML mining . . . 4
- 1.3 Research assumptions . . . 6
- 1.4 Research objectives . . . 7
- 1.5 Thesis structure . . . 8

2 XML Mining 11
- 2.1 Representing XML documents . . . 11
- 2.2 Measuring similarity . . . 16
- 2.3 Clustering . . . 21
- 2.4 Classification . . . 25
- 2.5 Performance evaluation . . . 27
- 2.6 Conclusions . . . 30

3 State of the art 33
- 3.1 Clustering . . . 33
- 3.2 Classification . . . 36
- 3.3 Approximate subtree matching . . . 38
- 3.4 Conclusions . . . 39

4 XPattern — a framework for clustering XML data by patterns 41
- 4.1 Conceptual description . . . 41
- 4.2 Formal definition . . . 43
- 4.3 Generic algorithm . . . 45
- 4.4 Conclusions . . . 45

5 XCleaner2 — a tree-based instance of the XPattern framework 47
- 5.1 Algorithm . . . 47

5.2 Example . . . 50
5.3 Experimental evaluation . . . 52
5.3.1 Datasets and experimental setup . . . 52
5.3.2 Parametrization . . . 53
5.3.3 Comparative analysis . . . 53
5.4 Conclusions . . . 54

6 PathXP — a path-based instance of the XPattern framework 57
6.1 Algorithm . . . 57
6.2 Example . . . 60
6.3 Parametrization . . . 62
6.4 Experimental evaluation . . . 65
6.4.1 Datasets and experimental setup . . . 65
6.4.2 Alternative pattern definitions . . . 66
6.4.3 Component analysis . . . 68
6.4.4 Parametrization . . . 71
6.4.5 Comparative analysis . . . 75
6.5 Conclusions . . . 76

7 Partial tree-edit distance 77
7.1 Preliminaries . . . 78
7.2 Conceptual description . . . 79
7.3 Formal definition . . . 80
7.4 Dynamic algorithm . . . 84
7.5 Experimental evaluation . . . 85
7.5.1 Datasets and experimental setup . . . 86
7.5.2 Combining PTED with a rule-based classifier . . . 86
7.6 Conclusions . . . 87

8 K-nearest patterns algorithm for pattern-based XML classification 89
8.1 Training . . . 89
8.2 Classification . . . 92
8.3 Example . . . 94
8.4 Experimental evaluation . . . 96
8.4.1 Datasets and experimental setup . . . 96
8.4.2 Component analysis . . . 97
8.4.3 Parametrization . . . 101
8.4.4 Comparative analysis . . . 103
8.5 Conclusions . . . 104

9 Final conclusions and future work 107

A Subtree Matching Algorithm 113

Bibliography 117

Index 129

1 Introduction

In year 2012, there was around 2.5 EB of data created every day [LXJZ14]. This rate has grown so rapidly, that 90% of data in the world today has been produced in the last two years. Such a vast amount of information poses an important challenge to the data processing community — a challenge commonly referred to as big data.

Usually, when discussing big data, three main components are distinguished: volume, velocity, and variety. Volume concerns both the rate at which data is created and its amount, e.g., 12 TB of Tweets are created every day. Velocity adds a time constraint on the processing of data, e.g., 5 million trade events have to be analyzed on the fly every day in order to detect potentially fraudulent transactions. The final dimension of big data is the variety of data types caused by the diversity of domains of their origin. Database records, text documents, image collections, social networks, websites, graphs, sensor data, are only a few types which require dedicated processing methods. Given that big data contains information of potentially high value, there is a strong need to address these challenges and design automated methods for discovering knowledge in such environments.

Knowledge discovery is a complex process of converting raw data into useful information. It consists of several steps, such as data selection, preprocessing, transformation, data mining, and evaluation, all of which have grown to constitute separate research areas. Among these, the core task is data mining, where statistical models, machine learning algorithms, and other methods are applied in order to discover previously unseen and nontrivial patterns in data. Because of the variety of data types, these methods often need to be designed specifically for a particular format. One of the most prominently used data formats across various domains is XML.

XML (eXtensible Markup Language) became an official W3C [W3C95] recommendation in 1998 [BPSM98]. Its main purpose was to fill the gap between simple, presentation-oriented HTML and complex SGML. It was designed specifically for exchanging data over the Internet in both human- and machine-readable form, while also being general enough to remain domain and platform indepen-

dent. Since then, it has gained great popularity in nearly every domain which involves exchanging information. Its applicability spans from data integration and collecting sensor data, to encoding chemical compounds and mathematical equations.

XML document is a data structure, in which named elements are linked together to form hierarchies of metadata which contain the actual data. The metadata, i.e., elements, attributes, and relations between them, form the *structure* of a document, while the actual data forms its *content*. Consequently, XML processing can be carried out according to three different scenarios: analyzing only the content of documents, using both content and structure, or focusing solely on the structure. In the first approach, well established text mining methods can be applied [SW03, KSC02]. The two latter cases require dedicated methods for dealing with the structure.

Structure of XML documents can be modeled in many different ways, depending on the required information granularity and efficiency of the process. Generally, the larger the information pieces, the more time-consuming the solution. Since the structure of XML documents forms a hierarchy, the most natural and complete representation for such data is a tree, as it allows for modeling all the information in the documents. However, tree processing is very costly and methods based on this representation can be unfeasible when larger datasets are to be analyzed. Moreover, some applications require a certain level of generality, in which cases using too detailed representation can lead to unexpected results. In such cases, when faster and more general methods are needed, representations based on document decomposition can be used. Depending on the necessity, such a decomposition can be accompanied by either a slight information loss (e.g., decomposing a tree into a set of subtrees), a significant information reduction (e.g., representing a document by the number of its elements), or anything in between. In general, choosing a particular representation will have a substantial impact on the result of the applied method and, therefore, even though is one of the first decisions to make, should be carried out carefully, with the expected end result in mind. This dissertation focuses on the structural information in XML documents and discusses it in the context of two core data mining tasks: clustering and classification.

Clustering is a data mining technique which aims at grouping together similar objects. Since there is no ground truth about the dataset and the analysis is performed solely based on the intrinsic characteristics of the data, clustering belongs to the category of unsupervised learning problems. In classification, on the other hand, the goal is to predict classes of new, previously unseen data, based on the model created during the training process. In order to train a classifier, classification process requires the knowledge about the ground truth, so it belongs to the category of supervised learning problems.

In the context of XML, both clustering and classification have gained much attention in the data mining research community and have been intensively studied during the last several years [PBML14, AMNS11, But04, TCY09]. Such an interest in this particular area can be attributed to the popularity of the XML

format and wide applicability of these methods across many different domains. The use of these methods spans from such domains as biology — to find groups of similar genes, through geography — to identify tourist travel patterns, to data integration — to automatically identify data sources. Let us now exhibit the relevance of the discussed subject by presenting a more detailed overview of the applications of structural clustering and classification of XML data.

1.1 Applications of XML clustering and classification

Data integration

Thanks to its flexibility and human-readable form, XML is one of the most popular formats for exchanging data and making it available through the Internet. If one wishes to integrate such data it is essential to translate the documents into a common schema to identify the corresponding pieces of information. Identifying different data sources is a good way to start since documents originating from the same source are likely to share a common schema. This can be achieved by XML clustering [VMB08], where different pieces of the same information encapsulated in the same structures can be found. Afterwards, documents can be integrated at the semantic level.

Query processing

When storing large collections of XML documents, their physical arrangement can play an important part in the efficiency of their later retrieval. One way of achieving such an arrangement is to cluster the documents prior to them being saved. Later, when the database is queried, accessing the data on the cluster level can reduce the query execution time by skipping the irrelevant parts of the dataset. Such a work has been performed for path queries and was shown to improve the native XML data storage [CMK07].

Web mining

Clustering of XML documents may be used not only before the documents are stored (like in the query processing example) but also on demand or to create indexes, in cases when we have no influence on their physical arrangement. A good example of such a case is web mining, where clustering can be performed to discover groups of resources that have a related content [VPD04]. Since XML is a popular format for data annotation, clustering does not even have to be restricted to web pages. Images, movies, user sessions on web servers, all can be included in a comprehensive web search, given they are properly annotated.

DTD extraction

Several tools for extracting DTDs from sets of XML data are in use. These tools generate a single schema based on a given document collection. Therefore, if the dataset consists of documents of several or unknown origins, it is imperative to identify structurally similar groups within these documents prior to schema extraction. This way, it is possible to find the actual DTDs on which the documents are based, rather than creating a new schema reflecting the whole document collection. This task can be achieved using structural XML clustering.

Bioinformatics

Structural XML clustering can also find various applications in many bioinformatics tasks. One of such tasks is finding sets of proteins sharing a similar structure [AAWS09]. A different example of biological application of XML clustering is gene clustering. Similarities among gene structures can account for common functions or a common transcriptional mechanism. A relatively high applicability of XML mining methods in bioinformatics can be attributed to a large collection of available standards based on the XML format, e.g., BSML [BSM97], ProML [HZL02], PDBML [WIN+05], SBML [HFS+03], CML [CML95].

Spatial data management

Another domain which benefits from the solutions developed within the field of XML clustering is spatial data management. Thanks to the popular geographical XML standard GML [Pos09] and the ability of modern cameras to automatically tag photos according to their location, XML clustering can be performed for several reasons. Collections of GML encoded areas can be grouped in order to identify similarities among regions, e.g., forests with lakes [ZJS10]. Geo-tagged photos, on the other hand, can help identify popular tourist travel patterns [ZLZC11]. Such a knowledge is of high value for touristic industries and would be difficult and expensive to obtain otherwise.

1.2 Open challenges in XML mining

Even though XML clustering and classification have been extensively studied throughout the last several years, there still remain some open issues within the developed solutions. Some of those issues are generic and relate to XML mining in general, some are specific to either clustering or classification, while others concern only particular approaches. In the next few paragraphs we will briefly discuss the main problems with the existing solutions.

Information type

One of the factors that influence the result of data mining algorithms the most is the type of information they use. Two cases can be considered here: using local or global information. *Local information* is obtained by analyzing only a fraction of the dataset through direct object similarity. In general, approaches using this information rely on comparing all objects with each other and making a decision about each of them based on the most similar object. In the case of clustering and classification, this decision concerns cluster and class assignment, respectively. On the other hand, *global information* is obtained through analyzing the global characteristics of the dataset. Approaches using this information usually rely on extracting common features among the objects in the dataset and making a decision about each of them based on the features they contain.

In XML clustering, nearly all of the existing methods are based on the same, local-information-oriented framework [AMNS11]. As a result, these methods are often sensitive to individual characteristics of particular documents and do not capture the global characteristics of the dataset. Moreover, no such framework is available for approaches relying on global information. In XML classification, on the other hand, we can observe more algorithms relying on global information. However, such classifiers often manifest poor dataset coverage or make predictions based on arbitrary calls because their classification model is too general. As it is not always clear which type to choose, designing a flexible method able to blend both types of information would be highly desirable.

Interpretability

Data mining is an interactive process and as such should not only give an answer to the stated problem, but also provide an insight into the underlying structure of the solution and allow for better understanding of the analyzed data. Many methods miss out on this important feature, providing only an answer but lacking in the interpretability of the obtained result. In clustering, each group is usually described by a single value (e.g., k-means) which does not reflect the structural characteristics of the documents, by a single document (e.g., k-medoids) which often displays individual rather than common features, or is not described at all (e.g., agglomerative hierarchical clustering). Even though this drawback can be attributed mainly to clustering, some classification algorithms also have this property (e.g., k-nearest neighbors). That is why, generating self-descriptive and easily interpretable results is a desirable feature of a useful data mining solution.

Parametrization

When applying any data mining algorithm to a real problem, another critical issue arises — what parameters should be used to achieve the best results. In classification, when the ground truth is given and a model can be tested, tuning the parameters can be performed in a methodical manner. However, in clustering, where no particular outcome is expected, this issue becomes more relevant and

difficult to approach. In particular, most clustering algorithms require that the number of clusters is given a priori [PBML14]. However, such a requirement can disqualify an algorithm in many real-world scenarios where no prior knowledge about the underlying structure of the dataset is given. That is why, it is essential to develop parameterless approaches or at least provide a good approximation for the initial parameter values.

Data distribution

A separate subject that needs to be considered when either developing or applying a data mining algorithm is data distribution. This issue is particularly problematic since highly imbalanced datasets can significantly distort the outcome of an algorithm yet remain undetected by many evaluation measures. This issue can also be attributed to both of the discussed data mining methods. In clustering, often large groups of objects tend to absorb smaller ones, while in classification, it is generally more favorable to assign objects to the majority class in terms of the average accuracy. Therefore, data distribution is another important issue to consider when designing a new data mining method.

Data homogeneity

Another important factor that should be taken into account, particularly when choosing a representation for XML documents, is the character of the dataset to be analyzed. Two cases can be considered here: documents originating from the same or from different sources, i.e., homogeneous and heterogeneous data, respectively. This distinction has a major influence on the problem's complexity. Documents originating from heterogeneous data sources are generally less difficult to analyze due to easily identifiable differences in tag labels. In such cases, lightweight document representations, such as tag or edge vectors, are likely to be sufficient. Homogeneous datasets, on the other hand, often share the same tag vocabulary. In such cases, more complex representations like paths or trees are more appropriate.

1.3 Research assumptions

Taking into account the popularity of XML format and a wide applicability of XML mining methods (as evidenced in Section 1.1), we argue that the problems stated in the previous section are relevant and worth pursuing. Let us summarize the most important issues which will allow us to formulate the main assumptions of this thesis.

Even though XML clustering methods based on global information exist, there is no formal methodology (framework) describing this process, whereas such a framework is available for local-information-based methods (see Section 2.3). Moreover, most of the existing approaches offer poor result interpretability, as

clusters constructed using direct document similarity do not summarize their content in an easily understandable way.

In the XML classification domain, the commonly used rule-based approach (defined around global information) faces an important issue, namely, what to do when a given document does not match any rule in the classification model. This commonly occurring case is usually dealt with a so called default rule (see Section 2.4). However, since this leads to a somewhat arbitrary class assignment, such a situation should be avoided. It can be addressed by using a kind of similarity measure able to provide the information about the degree in which a rule matches a given document. This information would allow to blend some local information into the global-information-oriented algorithm and could also help organizing the ranking of the rules. Such a measure, however, is unavailable for objects defined as tree structures, as is the case with XML documents.

Apart from balancing the use of local and global information in XML clustering and classification approaches, another common issue is parametrization. Extracting patterns from XML datasets requires defining a threshold of minimal number of documents in which a given piece of information (e.g., a subtree) needs to appear in order to call it a pattern. This parameter is highly dependent on the characteristics of a given dataset and usually difficult to acquire, especially in the case of clustering. Additionally, most of the existing clustering approaches require the number of clusters to be known a priori. However, such an assumption can discard the use of an algorithm in real-world applications, where the number of clusters is often unavailable.

Based on the above, we can formulate the main assumptions of the thesis:

- The existing XML clustering algorithms based on local information do not guarantee sufficiently interpretable results, what undermines their usefulness in real-world applications.
- There is no formal methodology for clustering XML documents by patterns which would systematically guarantee easily interpretable results.
- Classification of XML documents using the rule-based approach suffers from an unaddressed problem of default rule usage.
- There is no measure capable of accurately determining the degree of containment of one tree in another.

1.4 Research objectives

Given the above assumptions, the aim of this dissertation is to address the described issues by proposing new methods for XML clustering and classification based on the notion of patterns. Using patterns will allow for incorporating global information into the mining process, while introducing a pattern-document similarity measure should improve the dataset coverage and allow for blending in some

local information. The choice of proper parameter values for the proposed solutions should be experimentally evaluated and discussed. The work should also contain an in-depth analysis of several pattern definitions suitable for different dataset characteristics. Moreover, the results produced by the proposed methods should be self-descriptive and easy to interpret. Addressing the aforementioned problems should not diminish the quality of the achieved results, so the developed methods should produce results of quality at least as good as the current state-of-the-art approaches.

The main result in the XML clustering domain will be a pattern-based XML clustering framework and a working instance of this framework. The algorithm will be based on frequent paths and in addition to the main result will also provide an easily interpretable summary of each cluster. Moreover, a method for automatic number of clusters detection will be discussed.

The main result in the XML classification domain will be an algorithm based on both global information — expressed via patterns (subtrees) and local information — available through pattern-document similarity. In order to estimate the pattern-document similarity, a measure capable of assessing the degree of containment of one tree in another will be discussed. Furthermore, a dynamic programming algorithm, capable of calculating the measure efficiently, will be conceived.

The discussed methods will be experimentally evaluated in order to assess their quality, parameter sensitivity, and compare them against the current state-of-the-art approaches in their respective domains.

To sum up, the main objectives of this thesis are as follows.

- Proposing and formalizing a generic pattern-based framework for XML clustering.
- Validating the framework with a working example algorithm.
- Analyzing alternative pattern definitions in terms of their applicability to different types of datasets.
- Proposing and validating a pattern-based algorithm for XML classification.
- Proposing a measure for evaluating similarity between pattern trees and document trees.
- Analyzing the parametrization of the proposed approaches.

1.5 Thesis structure

The remainder of this thesis is organized as follows. In Chapter 2, we will present the core concepts regarding clustering and classification of XML data. First, we will discuss various document representations and applicable similarity measures. Next, we will introduce the basic notions regarding clustering an classification. Finally, we will analyze how to evaluate the performance of these methods. In the

end, we will present a sample dataset which will be used throughout the thesis to illustrate the proposed algorithms.

Chapter 3 will summarize the most relevant and characteristic algorithms in the discussed structural XML mining domains. The literature analysis will encompass the two main fields of inquiry, i.e., XML clustering and XML classification. Additionally, since this work also tackles the issue of subtree similarity, the approximate subtree matching area will be reviewed. By doing so, we will try to showcase that the problems stated in this work are still relevant.

Afterwards, we will dive into the first main field of study — XML clustering. In Chapter 4, we will discuss a new framework for clustering XML documents, called XPattern, which focuses on using global information expressed by patterns. We will begin by describing the intuition behind this approach, presenting its conceptual description, and comparing it with the existing local-information-oriented framework. After highlighting the main differences between the two approaches, XPattern will be formally defined.

In the two following chapters, we will discuss two example algorithms based on the proposed framework. Chapter 5 will discuss a tree-based instantiation of XPattern, called XCleaner2, which defines patterns as maximal frequent subtrees. We will describe how each component of the framework is specified in the defined algorithm and illustrate how it operates on a simple example. The proposed approach will be tested on both synthetic and real datasets for clustering quality and parameter sensitivity and compared against the state-of-the-art algorithm.

The second XPattern instance, called PathXP, will be discussed in Chapter 6. Analogously to XCleaner2, we will examine how each component of the framework is instantiated in the algorithm and illustrate its operation with the same clustering example. With the help of PathXP, we will also discuss possible heuristics for automatic detection of the number of clusters. In addition to the standard quality and parameter sensitivity tests, we will also experimentally evaluate what impact does each of the algorithm's components have on the clustering quality. Furthermore, we will explore several alternative pattern definitions and discuss their applicability to different types of datasets. The evaluation will be finalized with an experimental comparison of PathXP with its main competitors.

After presenting the XML clustering framework and its two instances, we will shift to the second field of study — XML classification. In Chapter 7, we will focus on the problem of default rule usage in the rule-based XML classifier by proposing a measure able to assess the degree of containment of one tree in another. First, the proposed measure, called partial tree-edit distance, will be conceptually described and formally defined. Afterwards, we will discuss a dynamic programming algorithm which will enable us to compute the measure in an efficient way. In this chapter, we will also illustrate the gravity of the default rule usage problem with a simple experiment and verify if partial tree-edit distance is able to address it.

In Chapter 8, we will extend the ideas from the previous chapters, i.e., patterns and pattern-document distance measure, into the XML classification domain by proposing a new pattern-based algorithm, called k-nearest patterns. In addi-

tion to the formal definition, the algorithm's operation will be illustrated with a simple example. Several variations regarding the algorithm's components will be discussed and experimentally evaluated. Finally, the proposed approach will be tested for quality and parameter sensitivity, and compared with the state-of-the-art algorithm.

After discussing the pattern-based XML classifier, we will arrive at final conclusions. In Chapter 9, we will summarize the thesis, confront its findings with the stated research objectives and draw lines of future research.

2 XML Mining

Mining in semistructural documents has been a hot topic in the last several years. As a result, numerous approaches and subdisciplines emerged from this research area. This chapter will summarize the most relevant findings in the XML mining domain. First, the two most important subproblems, common to all methods and applications, will be discussed: Section 2.1 will present the most popular representations for XML documents while Section 2.2 will discuss applicable measures for evaluating similarity between them. Afterwards, Sections 2.3 and 2.4 will focus on two data mining methods which gained the most attention in the XML mining research community, namely, clustering and classification. Finally, in Section 2.5, measures for evaluating the results of the applied methods will be analyzed.

2.1 Representing XML documents

Before applying any structural data processing task to a collection of XML documents, one needs to decide on how to represent the structure of the documents. This decision has a huge impact on the whole process as it determines the information available to other tasks after the transformation and, therefore, limits the possible processing methods to apply. That is why, it is very important to choose the right representation for a given problem. This section will summarize the most popular structural representations for XML documents and analyze their strengths and weaknesses.

XML structure

In order to analyze the available structural representations, first, we need to establish what the XML structure actually is and what information it contains. The most basic unit of any XML document is ***element***. It consists of a name, an ordered or unordered set of elements, and a set of attributes and textual val-

ues. ***Attribute*** is a key-value pair, where the key is the attribute's name and the value contains some textual information. ***XML document*** is a document containing exactly one main element which can contain other elements (subelements), attributes, etc.. The nesting of elements within a document is illustrated by an opening `<element_name>` and a closing `</element_name>` tag. Removing all textual values from an XML document reveals its structure, example of which is illustrated in Figure 2.1. In the presented structure, `inbook` is an example of an element while `year` is an example of an attribute.

```
<paper>
  <editors>
    <person />
    <person />
    <person />
  </editors >
  <publisher />
  <inbook year volume >
    <title />
  </inbook>
  <note />
</paper>
```

Figure 2.1: Example XML structure

The structure of XML documents carries two kinds of information which can be described as explicit and implicit. *Explicit information* consists of element and attribute ***labels***, i.e., their names. *Implicit information* is constituted by different types of relationships between the elements. Since the structure of an XML document forms a hierarchy, we can distinguish two types of relationships: *vertical*, i.e., relationships that occur along the document hierarchy, and *horizontal*, i.e., relationships that occur across the document hierarchy. The vertical relationships are: ***parent-child*** relationships between elements and their direct subelements/attributes, and ***ancestor-descendant*** relationships between elements and their direct or indirect subelements/attributes. The horizontal relationships are: ***sibling*** relationships between the subelements of the same element, and ***precedence*** relationships among siblings, constituted by their order. All of the described types of information are presented in Figure 2.2 under the *direct* category, as they can be obtained from XML documents without any additional processing. The *indirect* category represents the information obtained by processing the direct information (e.g., statistics, plots, etc.).

Different representations can use these information types to a varied extent, e.g., one can use a subset of direct information or create new information (indirect). The next subsections will present various structural XML representations which will use both of these strategies. Since all of the presented approaches treat attributes and elements equivalently, for convenience, hereinafter we will refer to both elements and attributes by ***elements***.

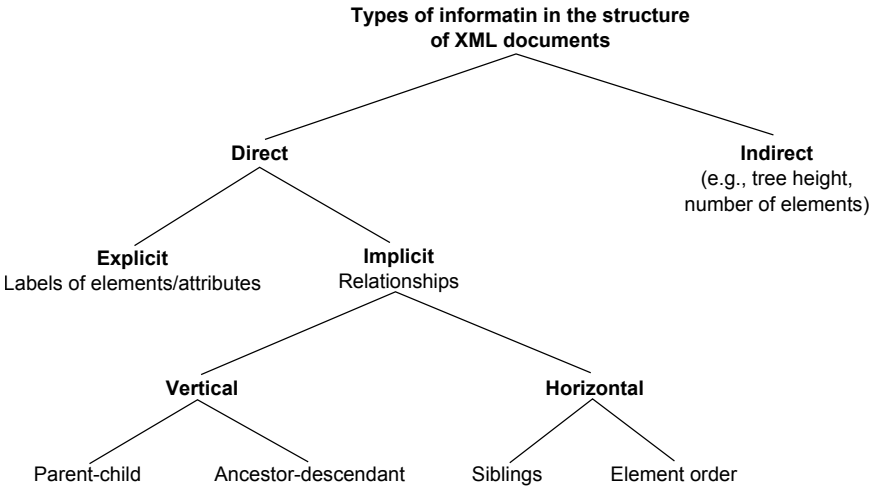


Figure 2.2: Information types available in the structure of XML documents.

Tree

The most natural representation for an XML document is a rooted, ordered, labeled tree, as it encapsulates all of the document's direct information. A ***tree*** t is a connected graph with $|t|$ nodes and $|t|-1$ edges. A tree t is ***rooted*** if all edges in t are directed away from one designated node, called ***root***. A rooted tree $|t|$ is ***ordered*** if there exists a total order among all nodes in t. The fact that a node x appears in a tree before a node y is expressed by $x < y$. A tree $|t|$ is ***labeled*** if every node x in this tree $x \in t$ has a label — symbolized by $l(x)$ — assigned to it from a finite alphabet. For convenience, hereinafter, a rooted, ordered, labeled tree will be referred to as ***tree***. Figure 2.3 illustrates the structure of the example XML document from Figure 2.1 represented as a tree.

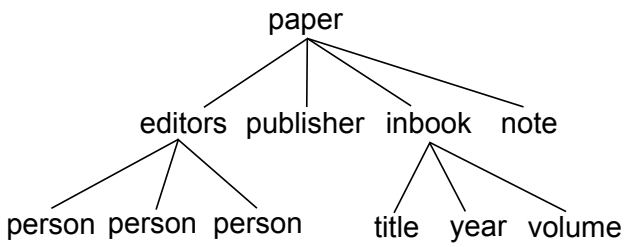


Figure 2.3: Structure of an XML document represented as a tree.

The main advantage of this representation is the fact that it carries all possible information encapsulated in XML documents. In this regard, it is the best one to choose when very detailed processing is required (e.g., when trying to identify differences within highly homogeneous data). However, when the emphasis is on efficiency, tree representation is usually too complex to process.

Set

In cases when both, complex processing and efficiency are required, a good solution is to represent an XML document as a decomposed tree in a form of a set

(or a multiset) of its parts. As trees can be decomposed to a varied extent, this representation can be flexibly adapted to a given problem when more information saturation or better performance is needed. Therefore, the decomposition can be drastic (e.g., set of labels, where only explicit information is preserved) or more subtle (e.g., set of subtrees, where only some relationships between the elements are lost). A tree s whose nodes and edges form subsets of nodes and edges of another tree t is called a ***subtree*** of t. The fact that s is a subtree of t will be symbolized by $s \subseteq t$.

It is worth noting, that there are other possible definitions of a subtree. The subtree defined as presented above is often referred to as ***induced subtree***. Alternatively, one can analyze embedded subtrees. A tree s is called an ***embedded subtree*** of a tree t if: i) nodes in s form a subset of nodes in t and ii) for any pair of nodes $x, y \in s$, if x is an ancestor of y in t then x is an ancestor of y in s, and iii) for any pair of nodes $x, y \in s$, if $x < y$ in t then $x < y$ in s. The difference between the two definitions is that induced subtrees preserve all information from the original tree, including parent-child relationships, while embedded subtrees do not preserve the parent-child relationships, only the ancestor-descendant. As this work focuses on induced subtrees, when there is no ambiguity, we will refer to induced subtrees by ***subtrees***.

Apart from labels and subtrees, a particularly popular decomposition unit is a path, or even more so — a ***full path***, i.e., a path running from the root of a document to one of its leafs. The example XML document from Figure 2.1 represented as a multiset of full paths is shown in Figure 2.4.

$$
\begin{aligned}
d = \{\ & \text{paper/editors/person},\\
& \text{paper/editors/person},\\
& \text{paper/editors/person},\\
& \text{paper/publisher},\\
& \text{paper/inbook/title},\\
& \text{paper/inbook/year},\\
& \text{paper/inbook/volume}\\
& \text{paper/note}\}
\end{aligned}
$$

Figure 2.4: Structure of an XML document represented as a multiset of full paths.

It is worth noting that even though the document is decomposed, complex structural processing can still be performed to the extent allowed by the chosen granularity level (e.g., a comparison of two documents represented as sets of paths can be performed down to the path-similarity level). Another important remark concerns the document's parts. It is very common to restrict the set representation to use only these parts which are frequent. A piece of information i is ***frequent***

in a dataset $\mathcal{D}$ if it appears in at least *minsup* percent of documents in $\mathcal{D}$:

$$frequent(i, \mathcal{D}) \Leftrightarrow \exists_{\mathcal{D}' \subseteq \mathcal{D}} \forall_{d' \in \mathcal{D}'} i \in d' \wedge \frac{|\mathcal{D}'|}{|\mathcal{D}|} \geq minsup, \tag{2.1}$$

where *minsup* is a user-defined minimum support parameter and $i \in d$ denotes that document d contains information i.

By adding the frequency restriction, we are reducing both the implicit and the explicit information. Consequently, there may be a need for a certain amount of preprocessing before transforming the documents into the set representation. Such is the case with frequent sets. It is thus important to realize, that this preprocessing may take a considerable amount of time, depending on the chosen decomposition level. For example, mining for frequent labels can be performed in linear time but mining for frequent subtrees yields exponential complexity.

Vector

Taking the information reduction a step further, one can decide to process XML documents encoded as vectors. In vector representation, similarly as in the set representation, a document is decomposed into several uniform pieces (e.g., labels, paths, subtrees). Here however, these pieces are encoded into a vector and only their presence or quantity is preserved. Concretely, each unique piece of document is assigned a position in a vector. The value at each position indicates either the presence(1)/absence(0) of the corresponding piece of the document or the quantity of that piece in the document. This representation is most commonly generalized to store the entire dataset as an $n \times m$ matrix, where n is the number of documents and m is the number of documents' parts unique across the whole dataset. The example document from Figure 8.1 represented as a binary vector encoding the presence/absence of document's labels is given in Table 2.1. It is important to note that the document is illustrated in the context of a larger dataset (labels `authors` and `journal` do not appear in the document). Otherwise, each value in the vector would be equal to 1.

Table 2.1: Structure of an XML document represented as a binary vector of labels.

Label	Presence
paper	1
editors	1
authors	0
person	1
publisher	1
inbook	1
journal	0
year	1
volume	1
title	1
note	1

Similarly as with the set representation, one may choose to limit the parts

encoded in a vector only to those which are frequent. In such a case, the same amount of preprocessing is required. However, unlike the set representation, after the encoding, the vector representation does not need any methods dedicated for XML or text in the further processing as simple linear algebra can be applied. On the other hand, this means that even though the processing will be faster, the results may not be as satisfying as with the set representation.

Another important remark is that the set representation is a generalization of the vector representation. In fact, encoding documents represented as sets into vectors is a common practice, however, in vector representation the information about the structures that correspond to the positions in the vector is lost while in the set representation, even encoded as a vector, this information is preserved.

Other

Of the representations covered so far, the tree and the vector are the most commonly used ones. However, there are other notable examples which we will now briefly describe. Taking the information reduction to the extreme level, one can choose to represent XML documents as single numbers summarizing the documents' structures. Such a representation completely neglects the direct information in the documents and focuses only on the indirect, computed information. Examples of such representations are: number of elements in a document, number of distinct elements, height of the document tree, average number of subelements of each element, etc.. These features also can be combined in the form of a vector. While such a representation most certainly is not suitable for more complex datasets, it is very efficient and can be used to cope with very simple data (see Section 6.4.2).

Among many other examples (compressed XML files [HAB12], elements with tree level information [Nay08], graphs [LCMY04]) probably the most creative one is the time series [FMM+05]. This representation relies solely on the depth of the elements and their order in the document. When reading an XML document element by element, each consecutive tag marks the time tick while the depth of the element it represents marks its value. This can be visualized by rotating an XML document by 90 degrees counterclockwise and drawing a line which follows its indentation. Figure 2.5 illustrates the process of transforming the example XML document from Figure 2.1 into the time series representation.

2.2 Measuring similarity

The most common XML processing task is similarity computation. It can be used either as a standalone technique, like in data integration, or as a part of a more complex process, e.g., clustering. That is why it has been the most studied subfield in the XML mining research community [But04, GMS07, TCY09]. As a result, there is a wide variety of measures available. However, as described in the previous section, similarity computation can be applied only after transforming

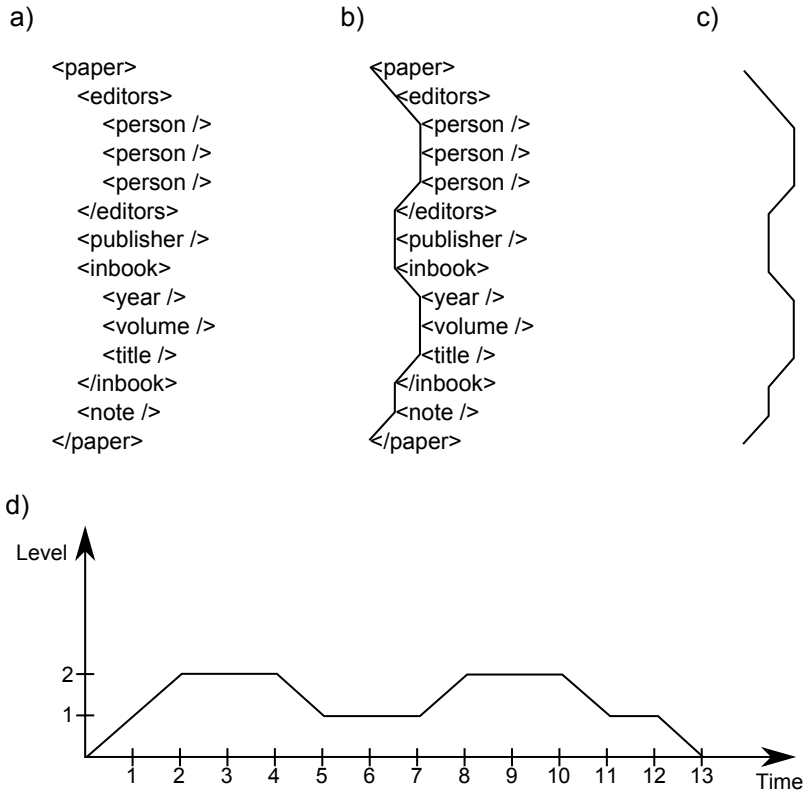


Figure 2.5: Process of creating a time series from an XML document.

the documents into a chosen representation. Consequently, the similarity can be measured only as accurately as the information delivered by the representation allows for. Since, usually, similarity computation is the actual task to perform, not the document transformation, a measure is chosen first and the choice of a compatible representation follows. This section will present the most popular structural XML similarity measures and match them with their corresponding representations.

Tree-edit distance

A measure which utilizes the full capabilities of the tree representation is tree-edit distance (TED). This measure was proposed long before the invention of XML and was dedicated for trees. The idea is based on a popular Levenshtein distance measure for sequences. A distance between two sequences can be computed as the minimal number of edit operations required to transform one sequence into another. These edit operations are: removing an element from the sequence; inserting an element into the sequence; renaming an element in the sequence. In the case of trees, the basic idea remains exactly the same, only the edit operations are redefined as follows. By ***inserting*** a node x into a tree t at a node y, x becomes a child of the parent of y, taking y's place in the sibling order, while y becomes a child of x. When ***deleting*** a node x from a tree t, all children of x become the children of the parent of x. ***Relabeling*** a node x to y means changing a label of x to $l(y)$.

Tree-edit sequence between two trees t_1 and t_2 is a sequence of insert, delete, and relabel operations which transforms t_1 into t_2. Assuming that each of these operations has an associated cost, the cost $c(s)$ of a tree-edit sequence s is the total cost of all operations in s. ***Tree-edit distance*** $\Delta(t_1, t_2)$ between two trees t_1 and t_2 is the minimal cost of all possible tree-edit sequences between t_1 and t_2.

$$\Delta(t_1, t_2) = min\{c(s) : s \text{ is a tree-edit sequence between } t_1 \text{ and } t_2\} \tag{2.2}$$

Equation 2.2 presents the basic form of the tree-edit distance measure which was proposed by Tai [Tai79] for trees. However, after XML was introduced, several other versions have been proposed. The first version dedicated for XML was put forward by Chawathe [Cha99]. In this measure, the insertion and deletion were restricted and could be performed only on the leaf nodes of a tree. Such a limitation was due to the fact that — as the author claimed — in the context of XML, inserting or deleting inner nodes is somewhat artificial because many relationships may be altered in the process. These restrictions highly reduced the complexity of the measure, allowing the author to propose an algorithm with an $O(|t_1||t_2|) = O(n^2)$ time complexity.

Another take on tree-edit distance was done by Nierman and Jagadish [NJ02]. The authors proposed a more complex yet more accurate version of the measure. In their solution, the insert and delete operations were also restricted, however, they could be performed on the whole subtrees[1]. The measure was tested on the clustering task and the proposed modification allowed the authors to achieve results of better quality than Chawathe.

Most recently, Wang et al. [WWZ+15] proposed a new version of tree-edit distance which adds two new tree operations: reverse and map. The reverse operation exchanges a parent node with one of its children while the map operation maps a path to an edge. The improvements proposed by the authors aim at facilitating the similarity join on XML documents.

The research concerning tree-edit distance can be conducted in two main areas: alternative measure definitions (like the ones presented in the previous paragraphs) or algorithm efficiency. Let us now focus on the second aspect. In order to present the basic tree-edit distance algorithm, we need to introduce some additional definitions.

A ***forest*** is an ordered set of trees. A tree t rooted at a node x is denoted by t_x while the root node of a tree t is denoted by r^t. A forest F containing trees rooted at all children nodes of a node x is denoted by F_x. The rightmost tree of a forest F is denoted by $\vec{F}$, a forest F without the rightmost tree is denoted by $F - \vec{F}$ and a forest F without the root of the rightmost tree is denoted by $F - r^{\vec{F}}$.

The basic formula for calculating tree-edit distance is given in Equations 2.3

[1] The authors used a different definition of a subtree than the one used in this dissertation. In their paper, the leafs of a subtree had to align with the leafs of the original tree.

and 2.4 and operates on two forests G and H:

$$\begin{aligned} \Delta(\emptyset, \emptyset) &= 0 \\ \Delta(G, \emptyset) &= \Delta\left(G - r^{\vec{G}}, \emptyset\right) + c_d(r^{\vec{G}}) \\ \Delta(\emptyset, H) &= \Delta\left(\emptyset, H - r^{\vec{H}}\right) + c_i(r^{\vec{H}}) \end{aligned} \tag{2.3}$$

$$\Delta(G, H) = \min \begin{cases} \Delta\left(G - r^{\vec{G}}, H\right) + c_d(r^{\vec{G}}) \\ \Delta\left(G, H - r^{\vec{H}}\right) + c_i(r^{\vec{H}}) \\ \Delta\left(G - \vec{G}, H - \vec{H}\right) + \Delta\left(F_{r^{\vec{G}}}, F_{r^{\vec{H}}}\right) + c_r(r^{\vec{G}}, r^{\vec{H}}) \end{cases} \tag{2.4}$$

where c_d, c_i, and c_r are the cost functions for deletion, insertion, and relabeling, respectively.

Equation 2.4 defines the main recursive formula of the measure. In each recursive step, three cases are considered:

1. Removing the root of the rightmost tree from the left forest.
2. Removing the root of the rightmost tree from the right forest (equivalent with inserting a new root node to the rightmost tree in the left forest.)
3. Relabeling the root of the rightmost tree in the left forest to the the root of the rightmost tree from the right forest.

Following these three steps recursively will eventually lead to meeting one of the boundary conditions from Equation 2.3. The first boundary condition states that an empty forest can be transformed into an empty forest at a zero cost while the second and the third conditions state that transforming any forest into an empty forest (or the other way around) is done at a cost of removing all of the nodes from that forest.

The presented formula was implemented by Zhang and Shasha[ZS89] with a dynamic programming algorithm. The authors noticed that there is no need to compute separate dynamic tables for all pairs of subtrees in both trees, as some of the results can be obtained as a byproduct. That is why, they used only a subset of subtrees rooted at the ***keyroots***.

$$keyroots(t) = \{r^t\} \cup \{x \in t : x \text{ has a left sibling}\}$$

Based on this fact, the authors were able to slightly reduce the complexity of the proposed algorithm to $O(|t_1||t_2||keyroots(t_1)||keyroots(t_2)|) = O(n^4)$.

In the algorithm proposed by Zhang and Shasha, the recursion always takes place on the right side of the forests. However, Klein [Kle98] found that altering the recursion direction between right and left sides reduces the number of the keyroots and, thus, the overall complexity. Concretely, the algorithm recurrences to the left if the size of the leftmost tree is less or equal to the size of the rightmost tree and to the right, otherwise. The author showed, that this allows to reduce the complexity of the dynamic algorithm to $O(|t_1|^2|t_2| \log |t_2|) = O(n^3 \log n)$.

Klein's idea was further improved by Demaine et al. [DMRW09] where the authors also alter between left- and right-side recursion. However, Klein only applies this strategy to the left forest which means that for example $\Delta(G, H)$ can be calculated faster than $\Delta(H, G)$. Demaine et al. addressed this issue and proposed an algorithm which has the optimal time complexity of $O(n^3)$.

Most recently, Pawlik and Augsten [PA11] proposed yet another improvement. Despite the fact that the $O(n^3)$ time complexity is optimal, the authors pointed out that the worst case actually takes place fairly often. Thus, the proposed improvement aims at matching the complexity and, additionally, assuring the choice of the optimal strategy when the worst case occurs.

Vector-based measures

The second most popular group of similarity measures is the one based on the vector representation. In this case, several measures are available, all of which are general and nonspecific to XML processing.

When two XML documents are encoded with binary vectors v_1 and v_2, the following measure can be applied:

$$Sim(v_1, v_2) = \frac{v_1^T v_2}{(v_1 \oplus v_2)^T (v_1 \oplus v_2)}$$

where $\oplus$ is a bitwise *or* operator. This measure illustrates the percentage of documents' parts which are common between the documents.

When two XML documents are encoded with vectors v_1 and v_2 representing real values (e.g., quantities of labels), the most popular measure is the Euclidean distance, defined as:

$$Dist(v_1, v_2) = \sqrt{\sum_{i=1}^{n} (v_1[i] - v_2[i])^2}$$

where $v_x[i]$ represents the i-th position in a vector v_x and n is the length of the vectors. This measure illustrates the geometric distance between the vectors in n-dimensional space.

The Euclidean distance is a popular choice thanks to its straightforward interpretation. However, the more dimensional the space, the more ambiguous the result. Furthermore, given that in the context of XML the values in the vectors often represent part quantities, penalizing the similarity equally for the differences in the quantities of the same parts as for the differences in the parts themselves can be inappropriate. In such cases, one can use the cosine distance, defined as:

$$Dist(v_1, v_2) = \frac{v_1 \cdot v_2}{||v_1|| ||v_2||}$$

It measures the angle between the vectors in n-dimensional space. This way, the information about the part quantities is preserved yet it has a smaller impact on the total outcome of the measure.

Other

Since trees and vectors are the most popular XML representations, as one would expect, tree- and vector-related similarity measures are among the most prevalent. Nevertheless, other representations also have their corresponding similarity measures. Examples of such measures are:

- normalized compression distance — for documents represented as compressed files [HAB12];
- weighted number of co-occurrences of the document parts at corresponding levels — for decomposed document with the information about the level of each part in the document tree [Nay08];
- modified discrete fourier transform — when encoding XML documents as time series [FMM+05];
- pq-gram distance — for documents represented as pq-grams [ABG05].

Looking at the described XML similarity measures one can notice a significant disproportion in the amount of research put into tree-edit distance and other areas. Several factors contribute to this situation. Firstly, tree-edit distance is not strictly an XML-specific problem and was analyzed long before XML was proposed (however, the same concerns vector-based measures). Secondly, tree-edit distance is the most accurate measure and as such attracts the most attention. Finally, as a consequence of its precision it is also largely complex and it took several research teams and many years to finally arrive at the optimal algorithm. In contrast, vector-based representations use well established and relatively simple similarity measures, so no new methods were necessary. As far as other representations are concerned, they are usually either much simpler than trees or are designed specifically to solve certain problems.

2.3 Clustering

Clustering is a machine learning technique which aims at partitioning a dataset of objects into subsets, called ***clusters***, which posses a desired property. It is easy to notice that this definition is rather "technically shallow" and only specifies that the subsets have to be created in a somewhat meaningful manner. Such a definition is clearly open to interpretation on what this "meaningfulness" implies. The most popular specialization of this definition states that clustering aims at creating clusters for which the inter-cluster object similarity is maximized and the intra-cluster object similarity is minimized. Most of the existing methods are designed around this definition. This section will describe the most popular clustering techniques used in the context of XML. It is worth noting that because clustering is defined around object similarity, most of these methods will be non-XML-specific. As a consequence, the XML-specific part is resolved by the applied similarity measure and its outcome can be fed to a generic algorithm.

Typically, the XML clustering task is divided into three main steps which form a general framework, presented in Figure 2.6. In the first step, the documents are transformed into a chosen representation, as described in Section 2.1. In the second step, the documents are compared with one another to determine their similarity according to a chosen measure. A description of XML similarity evaluation was given in Section 2.2. Finally, the similarity information is fed to a clustering algorithm which groups the documents accordingly. Additionally, the framework contains a feedback loop, which allows for similarity recalculation, as some algorithms use quality evaluation measures to refine the clusters. Applicable measures fur automatic quality evaluation as well as overall performance evaluation measures will be presented in Section 2.5.

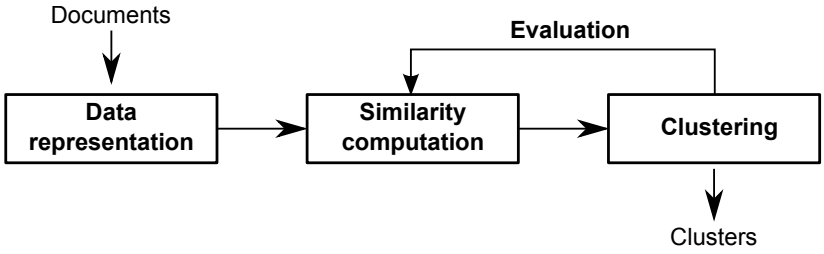


Figure 2.6: Traditional XML clustering framework.

Out of several different clustering techniques, two groups of approaches are used in the context of XML most frequently: hierarchical and partition-based methods. These algorithms are typically used as the third step of the general XML clustering framework presented in Figure 2.6.

Hierarchical

Hierarchical clustering algorithms generally appear in two variants: agglomerative (bottom-up) and divisive (top-down). In *agglomerative hierarchical clustering (AHC)*, sets of objects are iteratively grouped together, one pair at a time. The process starts with all objects forming separate clusters and ends with a single cluster containing all objects. At each intermediate step, the two most similar clusters are grouped together. As a result, we obtain a so called ***dendrogram*** — a structure depicting the order in which the clusters were formed (optionally with the similarity of each grouping). An example of such a dendrogram is given in Figure 2.7. If the desired number of clusters k is given by a user, we can simply select a cut-off point at the corresponding level in the dendrogram, as illustrated in Figure 2.7 with a red, dotted line. If, on the other hand, the number of clusters is unknown, we can manually or automatically select it analyzing the similarities used to group clusters in each iteration.

In contrast to the agglomerative algorithm, the *divisive* approach relies on iterative splitting of clusters. It begins with a single cluster containing all objects and ends with each object in a separate cluster. The division performed at each step optimizes a selected criterion, e.g., maximizes a distance between the resulting clusters, maximizes the Gini index, or minimizes the Entropy of the division.

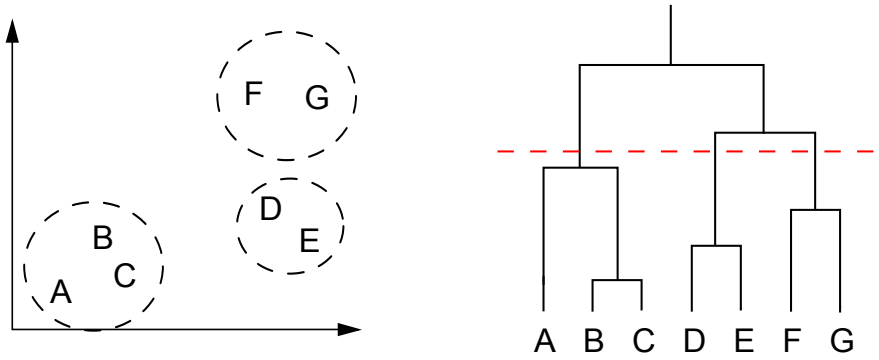


Figure 2.7: Example dendrogram illustrating the process of agglomerative hierarchical clustering.

Similarly to the agglomerative approach, as a result we obtain a dendrogram from which we can select the number of clusters analogously.

In both of the described variants of hierarchical clustering, there appears a common notion which needs to be specified, namely, cluster similarity. When two clusters contain a single object each, their similarity is simply evaluated using the distance between these objects. However, when more objects are present, there are several ways to deal with this issue. The most common solutions in this case are single-link, complete-link, and average-link similarity. In the *single-link* cluster similarity, the distance between two clusters is evaluated as the distance between the two closest objects in these clusters while in the *complete-link* approach, the inter-cluster distance is equal to the distance between the two furthest objects in the clusters. The first approach allows to find more oddly shaped clusters, however, is prone to the cluster chaining anomaly, to which the second approach is immune. A method which lies in between these two is the *average-link* approach, where the inter-cluster distance is evaluated as the average distance between all pairs of objects from both clusters.

Hierarchical clustering algorithms are generally prone to local optima, as the splits or joins of the clusters are irreversible and optimized at each step. The agglomerative approach is much more prone to this problem because it begins with local information about pairwise object similarity neglecting the global structure of the data. The divisive approach, on the other hand, starts with complete information about the global distribution, therefore, can produce more accurate results. This advantage, however, results in a much higher cost of the top-down strategy, which has a $O(2^n)$ time complexity, compared to a $O(n^3)$ complexity of the bottom-up approach. As a result, the agglomerative hierarchical clustering algorithm is considerably more popular than the divisive approach, also in the context of XML.

Partition-based

Partition-based clustering algorithms require the number of clusters k to be known a priori. They rely on splitting the dataset into k initial clusters and iteratively refining these clusters by relocating the objects between them. This procedure is

carried out until a stop condition is met, e.g., no objects are relocated between two consecutive iterations or a certain number of iterations is reached.

The most popular representative of the partition-based methods is the *k-means* algorithm. In this approach, the initial partitioning is based on k randomly chosen points in the object space, called ***centroids***, which mark the center points of clusters. All objects are assigned to the clusters based on their nearest centroids which are later recalculated as new center points of the clusters. Example execution of the k-means clustering is presented in Figure 2.8.

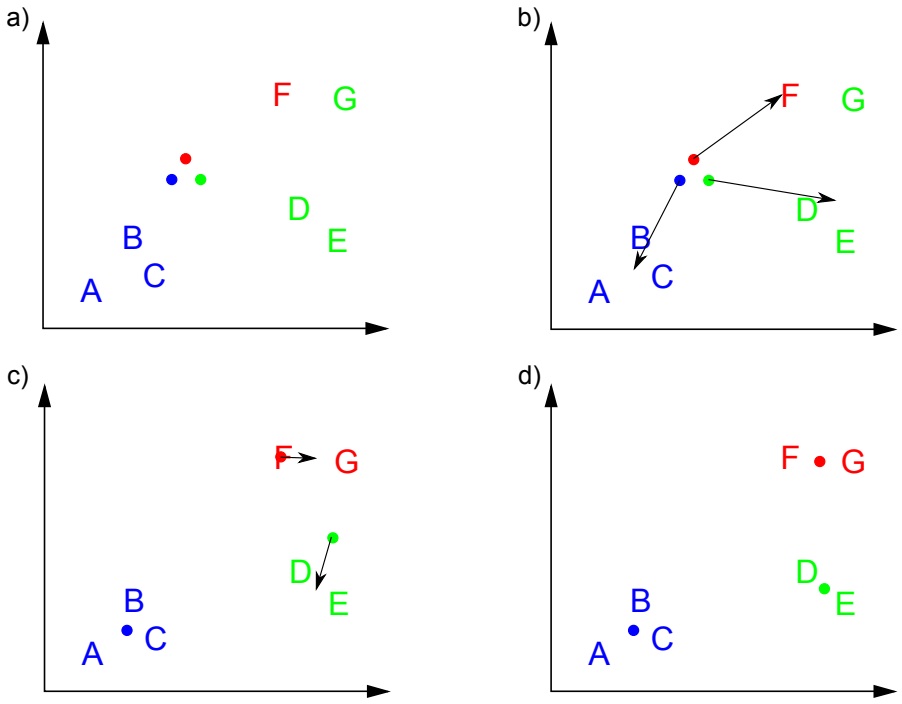


Figure 2.8: Example of the k-means algorithm execution.

It is worth noting, that even if the procedure naturally converges, there is no guarantee that the globally optimal solution has been found. Therefore, the algorithm is prone to local optima. Moreover, as the initial points are chosen at random, running k-means several times on the same dataset may result in different clusters. Finally, this approach is sensitive to outliers, as a single object far out in the object space can significantly shift the center of a cluster. This problem, however, can be solved using a variation of the k-means algorithm, called *k-medoids*. The basic procedure remains the same. The only difference lies in the definition of a centroid, which instead of being a center point of a cluster is an object nearest the center point of a cluster.

It is worth noting, that partition-based methods form flat results, as opposed to hierarchical approaches, which additionally provide the dendrogram. As a result, hierarchical approaches are more suitable in cases when the number of clusters is unknown and a deeper understanding of the underlying structure of the dataset is needed. Moreover, partition-based methods tend to form spherically shaped clusters because objects are grouped around a single center point. However, these methods are generally less complex than hierarchical approaches

(e.g., k-means has a $O(kni)$ time complexity, where n is the size of the dataset and i — the number of iterations). Finally, partition-based algorithms have the advantage of being able to relocate objects between clusters in each iteration. Such an action is impossible in hierarchical algorithms as cluster assignments are final. In this sense, these families of methods are somewhat complementary and can be chained together to compensate for each others' weaknesses. For example, one can start with an agglomerative algorithm to form k initial clusters and next, apply several iterations of k-means to refine the result. Another approach would be to first run a partition-based algorithm for $k' > k$ to form small and highly cohesive groups of objects and later group them to form k final clusters using AHC. A similar approach was proposed for XML clustering where the authors first form $k' > k$ clusters iteratively and later join them into k final groups in a hierarchical fashion [TNB07, KTNL07].

2.4 Classification

Classification is a machine learning task which aims at predicting classes of new, previously unseen objects, based on previous observations for which the classes are known. These objects which help to make the decision are called ***training examples*** and all together constitute a ***training dataset***. They are also frequently referred to as ***labeled*** data, as opposed to the objects for which the classes are unknown, called ***unlabeled***. The class prediction of unlabeled data is performed by a ***classification model*** — also called a ***classifier*** — which is created based on training data in a process called ***training***. In classification, the target value of the prediction is a discrete set of classes. Prediction of a continuous target value is performed by regression and is out of the scope of this dissertation.

Classification is a much less popular field than clustering in the context of XML processing. Although no formal XML classification framework is defined, the previously described representations (Section 2.1) as well as some similarity measures (Section 2.2) are still widely applicable. Similarly to clustering, the algorithms used for classification are mostly adoptions of general techniques. Below, the two most characteristic examples of classification methods used in the context of XML will be presented, namely, rule-based and nearest neighbor classifier.

Rule-based classifier

In *rule-based classification*, the classification model is composed of rules arranged into a ranking. Each rule has the following structure: $F \rightarrow c$, where F is a set of features and c is a class. In market basket analysis, where rule-based classification originates, features are defined as frequent items appearing in the baskets (transactions). In general however, they can be any pieces of information obtainable from the analyzed objects which are frequent across the dataset (see Equation 2.1). For example, in the context of XML they can be frequent labels, paths, or subtrees. After obtaining the rules, they are ordered according to the

precedence relation $\prec$. Before defining this relation, we need to introduce some additional definitions.

Support of a rule $F \to c$ is measured as a frequency of the feature set F within the training examples with class c.

$$supp(F \to c) = \frac{|\{o \in \mathcal{D}_c : \forall_{f \in F} f \in o\}|}{|\mathcal{D}_c|},$$

where $\mathcal{D}_c$ are the training examples with class c. ***Confidence*** of a rule $F \to c$ is measured as a relative support of the rule in its class to the support of this rule in all classes in the training dataset.

$$conf(F \to c) = \frac{supp(F \to c)}{\sum_{c' \in \mathcal{C}} supp(F \to c')}$$

Size of a rule $F \to c$ is measured as a number of features in the feature set F.

$$size(F \to c) = |F|$$

Based on the above, rule r_1 precedes rule r_2, symbolized as $r_1 \prec r_2$:

1. if $conf(r_1) > conf(r_2)$
2. else if $conf(r_1) = conf(r_2) \wedge supp(r_1) > supp(r_2)$
3. else if $conf(r_1) = conf(r_2) \wedge supp(r_1) = supp(r_2) \wedge size(r_1) > size(r_2)$

Once the ranking is created, classification is performed as follows. Each new, unlabeled object is assigned to the class indicated by the first rule (according to the ranking) for which the object contains all of the features:

Rule-based classifier benefits from using the global information, i.e., information obtained by analyzing the whole dataset (see Section 1.2). This allows the algorithm to ignore the individual characteristics of each object and focus on the features which are common in a group of objects with a specific class. This is particularly useful in the XML classification scenario, as the feature set (elements, attributes, relationships) is not completely defined and new features may appear after the classifier is trained. However, concentrating only on global information can be problematic, especially in the described scenario of new features appearing after the training. If a rule-based classifier is asked for a class prediction of a previously unseen object with many new features or with features which were not common in the training dataset, there may not be any rule matching this object. In such cases, classifier makes predictions based on a ***default rule***, i.e., a special rule which always points to a single class, regardless of the features present in the classified object. The most commonly adopted strategy in this scenario is to point to the majority class in the training dataset. Based on the above, it is clear that this assignment is somewhat arbitrary and should be avoided.

Despite the default rule problem, the rule-based classifier is a good and intuitive solution for XML classification task. Thanks to a compact model, it is easy to analyze why each object was assigned to a certain class. Moreover, the model can also give an insight into the underlying structure of the training dataset.

Nearest neighbor classifier

An approach contrasting with the rule-based algorithm is the *k-nearest neighbors classifier (kNN)*. In kNN, the classification model is composed of all documents in the training dataset. The class prediction of a new object o_x is performed by selecting the k most similar objects from the training dataset, called k nearest neighbors, to the object being classified and assigning o_x to the majority class among those k objects. Formally, having an unlabeled object o_x and a set of its k nearest neighbors given as follows:

$$kNN(o_x, \mathcal{D}) = \mathcal{D}_k \subseteq \mathcal{D} : |\mathcal{D}_k| = k \wedge \forall_{o \in \mathcal{D}_k} \left[Sim(o_x, o) \geq \max_{o' \in \mathcal{D}/\mathcal{D}_k} \left(Sim(o', o) \right) \right]$$

where $Sim(o_1, o_2)$ is some similarity measure between objects o_1 and o_2, the class prediction c_x for object o_x is performed as follows:

$$c_x = \arg\max_{c \in \mathcal{D}} \left(|kNN(o_x, \mathcal{D})^c| \right)$$

where $kNN(o_x, \mathcal{D})^c$ is a subset of the k nearest neighbors of o_x with class c.

The kNN algorithm benefits from using local information, i.e., information available by analyzing pairwise similarity between objects in the dataset (see Section 1.2). By doing so, it is able to make an informed prediction regarding any object without the necessity of using a default class, as was the case with the rule-based method. Furthermore, no training phase is required as all training examples form a classification model. Such methods are often referred to as *lazy*. The laziness of kNN, however, translates to a costly prediction, because, theoretically, finding the nearest neighbor requires measuring similarity with all objects in the dataset. This problem, however, can be solved by using a distance metric or an index over the training data, both of which allow to find the nearest neighbors much more efficiently.

The use of local information also has its downsides. Firstly, the algorithm misses out on the global characteristics of the data what makes is susceptible to outliers. Secondly, the algorithm is sensitive w.r.t. the k parameter (which is user-defined) especially when dealing with homogeneous datasets, which is often the case with XML classification, where similarity is measured with tree-edit distance.

2.5 Performance evaluation

After performing a desired task, either clustering or training a classifier, the resulting model should be validated in order to verify if it is of acceptable quality and if further processing is needed. In case of classification, this is carried out using external measures, i.e., measures using external knowledge provided by labeled test datasets or via cross-validation on the training data. In clustering, labeled datasets are not always available. In such cases, internal measures, i.e., measures

relying solely on internal properties of objects without the external knowledge, have to be used.

External measures

External measures (also called *external indices* or *supervised measures*) verify how well does the constructed model reflect the ground truth given in test data. These measures use labeled datasets which contain objects already assigned to classes or clusters and evaluate how accurate the model is based on this information.

The first set of measures treats each class/cluster assignment as a separate decision made about each object. In such cases, a ***true positive (TP)*** decision correctly assigns an object to the positive class or two similar objects to the same cluster, whereas a ***true negative (TN)*** decision correctly assigns an object to the negative class or two dissimilar objects to different clusters. Analogously, ***false positive (FP)*** and ***false negative (FN)*** decisions incorrectly assign an object to the positive class or two similar objects to different clusters and an object to the negative class or two dissimilar objects to the same cluster, respectively.

Many measures are designed based on these four simple notions. Arguably, the most popular ones are Precision and Recall, which are somewhat complementary. ***Precision*** describes how many of the positively marked cases were actually positive while ***Recall*** describes how many of the positive cases were positively marked:

$$Precision = \frac{TP}{TP + FP},$$

$$Recall = \frac{TP}{TP + FN}.$$

In order to balance the penalization between between false positive and false negative decisions in a single measure, the ***F-score*** [BR99] indicy, which penalizes false negatives according to a user-specified factor $\beta > 0$, can be used:

$$F_\beta = \frac{(\beta^2 + 1)PrecisionRecall}{\beta^2 Precision + Recall}.$$

Specifically in classification, two other measures are commonly used, namely, Sensitivity and Specificity. ***Sensitivity*** is defined exactly the same as Precision, so describes how well the classifier recognizes positive class, while ***Specificity*** describes how well it recognizes the negative class:

$$Sensitivity = \frac{TP}{TP + FP},$$

$$Specificity = \frac{TN}{TN + FN}.$$

Another measure typically used in classification is ***Accuracy***:

$$Accuracy = \frac{|\mathcal{D}^{test}|}{|\mathcal{D}|} \tag{2.5}$$

where $\mathcal{D}^{test}$ is the set of correctly classified documents and $\mathcal{D}$ is the set of all documents. This version of Accuracy, also referred to as ***proportional***, favors larger classes. To compensate for this fact when class imbalance occurs (see Section 1.2), an ***equal*** variation is recommended:

$$Accuracy_e = \sum_{c \in \mathcal{C}} \left(\frac{1}{|\mathcal{C}|} \cdot \frac{|\mathcal{D}_c^{test}|}{|\mathcal{D}_c|} \right) \tag{2.6}$$

where $\mathcal{D}_c^{test}$ is the set of documents correctly assigned to class c and $\mathcal{D}_c$ is the set of all documents from class c.

One of the simplest external indices dedicated for clustering is the ***Purity*** measure [ZK02], which evaluates the degree in which a cluster contains documents from a single category. For a given cluster c_i and a set of possible clusters $\mathcal{C}$, the Purity of c_i is calculated as:

$$Purity(c_i) = \frac{1}{|c_i|} \max_{c \in \mathcal{C}} (c_i^c)$$

where c_i^c represents the number of documents from cluster c_i assigned to category c. The overall Purity of a clustering is defined as:

$$Purity(\mathcal{C}) = \sum_{c \in \mathcal{C}} \frac{|c|}{|\mathcal{D}|} Purity(c).$$

Furthermore, when interpreting each cluster assignment as a separate decision, the ***Rand Index (RI)*** [Ran71] was proposed as:

$$RI = \frac{TP + TN}{TP + FP + FN + TN}.$$

Internal measures

Internal measures (also called *internal indices* or *unsupervised measures*) measure the quality of clusters without the use of any external information about the way in which the analyzed objects should be clustered. Unsupervised measures are divided into measures of cluster *cohesion*, which determine the compactness of objects within a cluster, and cluster *isolation*, which determine how well a cluster is separated from other clusters [TSK05].

One of the most popular internal indices is the ***Sum of squared errors (SSE)***, a cohesion measure based on some distance metric $dist()$, calculated as:

$$SSE = \sum_{c \in \mathcal{C}} \frac{1}{2|c|} \sum_{o_x \in c} \sum_{o_y \in c} dist(o_x, o_y)^2.$$

The most popular separation measure is the ***Between group sum of squares (SSB)***, calculated as the sum of squared distances of cluster centroids c_c to the

overall mean $\tilde{c}$ of all the objects:

$$SSB = \sum_{c_c \in \mathcal{C}} |c_c| dist(c_c, \tilde{c})^2.$$

Cohesion and separation are often combined to ensure high intra-cluster and low inter-cluster similarity. An example of such a combination is the ***Silhouette coefficient*** [Rou87]. For an object o, the Silhouette coefficient is computed as:

$$Silhouette(o) = \frac{b(o) - a(o)}{\max\{a(o), b(o)\}},$$

where $a(o)$ is the average dissimilarity of o with all other data within the same cluster and $b(o)$ is the minimal of average distances between o and any cluster that does not contain o. The value of the Silhouette coefficient varies between -1 and 1, with 1 being the most desirable value.

External indices provide a transparent comparison between the algorithm's output and the desired outcome. Moreover, they allow to differentiate the importance of false positive and false negative errors. Unfortunately, in many clustering scenarios labeled datasets are unavailable. For this reason, clustering algorithms designed to tackle large datasets will most probably require unsupervised validation measures. In the field of XML mining by structure, the most popular universal evaluation methods are Precision and Recall. Additionally, Accuracy is commonly used in classification while SSE and SSB are used in real-world clustering, when the analyzed documents are unlabeled.

2.6 Conclusions

In this chapter, we have presented the core concepts regarding clustering and classification of XML data. We discussed various document representations and applicable similarity measures, introduced basic notions regarding clustering an classification, and tackled the issue of performance evaluation. In the next chapter, we will examine the state-of-the-art methods for XML clustering, classification, and approximate subtree matching.

When illustrating how the algorithms discussed in the further chapters of the thesis operate, we will use a sample dataset consisting of 8 XML documents presented in Table 2.2. Documents $d_1 - d_4$ correspond with DTD_1 and represent a class of book chapters c_1, while documents $d_5 - d_8$ correspond with DTD_2 and represent a class of journal papers c_2. This dataset will be used when discussing both clustering and classification. To better illustrate the tackled problems, the DTDs of the example documents are provided in Table 2.3. In reality, DTDs are usually unavailable during the clustering or classification process so it is important to stress that all of the approaches which will be described in the further chapters do not depend on DTDs. They are provided solely for illustrative purposes.

Table 2.2: Example training dataset created based on the DTDs from Table 2.3.

DTD_1 — class c_1			
d_1	d_2	d_3	d_4
<paper> <authors> <person /> </authors > <publisher /> <inbook> <title /> <year /> <volume /> </inbook> </paper>	<paper> <authors> <person /> </authors > <publisher /> <inbook> <title /> </inbook> <note /> </paper>	<paper> <authors> <person /> <person /> </authors > <inbook> <title /> <number /> </inbook> </paper>	<paper> <editors> <person /> <person /> </editors > <inbook> <title /> <year /> <volume /> </inbook> </paper>
DTD_2 — class c_2			
d_5	d_6	d_7	d_8
<paper> <authors> <person /> </authors> <journal> <title /> <year /> <volume /> </journal> </paper>	<paper> <authors> <person /> </authors> <journal> <title /> <year /> </journal> <note /> </paper>	<paper> <authors> <person /> <person /> </authors> <journal> <title /> <year /> <volume /> </journal> <pages /> </paper>	<paper> <authors> <person /> <person /> </authors> <journal> <title /> <year /> <volume /> </journal> </paper>

Table 2.3: Example DTDs.

DTD_1	DTD_2
<!ELEMENT paper (authors|editors, publisher, inbook, note?, pages?)>	<!ELEMENT paper (authors, journal, note?, pages?)>
<!ELEMENT authors (person+)>	<!ELEMENT authors (person+)>
<!ELEMENT person (#PCDATA)>	<!ELEMENT person (#PCDATA)>
<!ELEMENT inbook (title, year?, volume?, number?)>	<!ELEMENT journal (title, year, volume, number?)>
<!ELEMENT title (#PCDATA)>	<!ELEMENT title (#PCDATA)>
<!ELEMENT year (#PCDATA)>	<!ELEMENT year (#PCDATA)>
<!ELEMENT volume (#PCDATA)>	<!ELEMENT volume (#PCDATA)>
<!ELEMENT number (#PCDATA)>	<!ELEMENT number (#PCDATA)>
<!ELEMENT note (#PCDATA)>	<!ELEMENT note (#PCDATA)>
<!ELEMENT pages (#PCDATA)>	<!ELEMENT pages (#PCDATA)>
<!ELEMENT editors (person+)>	
<!ELEMENT publisher (#PCDATA)>	

3 State of the art

The available document representations and similarity measures in conjunction with clustering and classification methods open many opportunities to create structural XML processing algorithms. Throughout the last years, several solutions utilizing the discussed methods have been proposed. This chapter will summarize the most relevant and characteristic algorithms in the discussed structural XML mining domains. By doing so, we will try to showcase that the problems stated in this work are still relevant.

It is worth noting, that there is a substantial disproportion in the quantity of available approaches, as XML clustering has been much more intensively studied than classification. This is due to the simple fact that XML clustering is much wider applied in practice, as evidenced in the introduction of this thesis. Apart from these two groups of algorithms, some of the approaches related to approximate subtree matching will also be reviewed, as further in the thesis, in the context of XML classification, a new measure designed to solve this problem will be proposed.

3.1 Clustering

According to our recent survey [PBML14], structural XML clustering algorithms can be categorized into four main groups: tree-edit distance, substructural similarity, level similarity, and other approaches. Below, the main approaches from each category will be summarized. It is worth noting, that nearly all of these algorithms share the clustering framework described in Section 2.3.

The first category is constituted by approaches relying on the tree-edit distance measure. Out of several algorithms based on this notion [Sel77, ZS89, CRGW96, Cha99] Nierman and Jagadish [NJ02] put forward an algorithm designed specifically for XML documents. As described in Section 2.2, the authors represent documents as trees and propose a slightly modified version of tree-edit distance

which allows for deleting or inserting whole subtrees. Using the proposed measure, the authors calculate distances between XML documents and cluster them using the agglomerative hierarchical algorithm (AHC, see Section 2.3). A similar approach was proposed by Tekli and Chbeirb [TC12], where subtree operations are also permitted. However, the subtrees found in the compared documents do not need to be identical. This way, when a subtree operation is performed, its cost is proportional to the similarity of the analyzed subtrees. The proposed measure was used with a single-link hierarchical method.

The tree-edit distance methods described so far operate on document trees in their complete, unmodified form. Dalamagas et al. [DCWS06] proposed to summarize XML trees into structures similar to dataguide [GW97]. The authors later compare the summarized documents using the tree-edit distance algorithm proposed by Chawate [Cha99] and cluster them with a single-link hierarchical method. Another approach based on processed trees was put forward by Yang et al. [YKT05]. The authors propose to transform each XML document into a structure called normalized binary tree, encode it in a vector and calculate the distance between two documents by subtracting the values at corresponding positions in their vectors. The authors proved that binary branch distance forms a lower bound for tree-edit distance. In contrast to tree-edit distance, the binary branch distance can be calculated in linear time and is one of the most efficient ways of comparing full tree structures.

The second category consists of methods based on tree decomposition. These methods offer a compromise between fast tag algorithms and accurate edit distance approaches. Interestingly, almost all of these approaches use path-oriented similarity measures. Vercoustre et al. [VFGL05] proposed several path-based representations. In each of them, paths are encoded into vectors of term frequencies scaled with TF-IDF [SB88] and clustered with the k-means algorithm (see Section 2.3), with centroids computed as a sum of all vectors in a given cluster and similarity evaluated using the Euclidean distance. Other approaches using quantity-based similarity include Leung et al. [LCCL05], Rafiei et al. [RMS06], and Costa et al. [CMOT04]. All three of these methods evaluate similarity using the number or percentage of common paths and cluster the documents using the AHC algorithm. The differences between them concern the paths they use: Leung et al. [LCCL05] use only maximal frequent full paths, Rafiei et al. [RMS06] use all paths starting at the root node, while Costa et al. [CMOT04] allow for any types of paths. Additionally, in the last algorithm, the inter-cluster similarity is evaluated using each cluster's representative (i.e., a tree which has the lowest distance from all trees in the cluster) while the two former approaches use single-link similarity.

A different approach was proposed by Lian et al. [LCMY04], where documents are summarized into graphs by removing repeating nodes and encoded as binary edge vectors. These vectors are later compared based on the percentage of common edges between them and clustered using the ROCK algorithm [GRS00]. Similarly, Aïtelhadj et al. [ABMS12] presented a method where documents are summarized by merging all repeating elements appearing at each tree level and

further decomposed into sets of full paths compared using a weighted sum of similarities between all of the paths. The authors, however, do not rely on any existing clustering algorithm but propose an original iterative procedure for constructing clusters. Each consecutive document is either assigned to one of the existing clusters if it satisfies a given similarity threshold with its representative, or constitutes a new one. The cluster representative is chosen as the document with the highest similarity with all other documents in this cluster. A very similar algorithm was proposed by Tran et al. [TNB07] and Kutty et al. [KTNL07]. The authors utilize a two-phase approach with each document represented as a set of full paths and a set of subtrees, respectively. The first phase is the same as in Aïtelhadj et al. [ABMS12] approach, however, the authors further merge the clusters according to their similarity, until they form the user-specified number of clusters. For similarity evaluation, Kutty et al. [KTNL07] use the Jaccard coefficient, while Tran et al. [TNB07] propose an original measure which calculates the joint similarity of all paths in both documents.

Finally, Aggarwal et al. [ATW+07] proposed an approach based on the notion of patterns defined as frequent edges. These patterns serve as centroids in the k-means algorithm used by the authors to cluster the documents. The initial partition is random and the distance between documents and centroids is defined as a fraction of common edges.

Methods in the third category take the tree decomposition one step further and use simple tag- and edge-based representations with the additional information about the level at which these substructures appear in the document tree. This approach was first proposed by Nayak [Nay08] and later enhanced by Alishahi et al. [ANA10]. In this method, each document is represented as a vector in which cells correspond to consecutive levels in the document tree. Each cell contains another vector encoding distinct tags which appear in the document at the corresponding level. Using this representation, documents are iteratively assigned to clusters which contain the most similar documents, where similarity is evaluated based on the weighted number of common tags at corresponding levels. A complementary approach was presented by Antonelli et al. [AMT08], where instead of tags, the authors use edges and encode them in the same level structure. However, in contrast to Nayak [Nay08] and Alishahi et al. [ANA10], the authors use the k-means algorithm, with centroids encoded with the same structure as documents — each level contains all distinct edges in the corresponding levels of all documents in a given cluster.

The final group consists of methods which cannot be unambiguously assigned to any particular category, as each of them presents a very unique approach. For example, Candillier et al. [CTT05] proposed yet another vector-based algorithm, however, the positions in this vector do not represent the same type of objects. Instead, they can represent the frequencies of parent-child relations, sibling relations, paths, tags, and absolute node positions. As such representations can grow fairly large, the authors propose to perform feature selection and cluster the documents using the Expectation-maximization algorithm [Moo96]. Another interesting method proposed by Hagenbuchner et al. [HST+05] is based on Ko-

honen's Self Organizing Maps (SOM) [Koh89]. In this approach, each document is mapped on the SOM neuron grid, one tree node at a time. The idea is that afterwards, similar documents should appear at close coordinates on the neuron grid and can be clustered accordingly.

Possibly, the two most creative approaches to dealing with the XML clustering task were presented by Flesca et al. [FMM+05] and Helmer et al. [Hel07, HAB12]. Both of these proposals are highly efficient and can be used with any similarity-based clustering algorithm. The authors of the first approach [FMM+05] propose to model XML documents as time series, where the depth of an element represents the strength of a signal in time given by the order in which it appears in the document (see Section 2.1). Given such a representation, documents are compared using the Discrete Fourier Transform. In the second approach [Hel07, HAB12], the structures of the documents are compressed with some file compression algorithm and the difference in the length of the compressed files serves as a distance measure.

As apparent in the abundance of approaches, clustering has been the most intensively studied subject in the context of XML. As a result, several reviews related to this task are available. The most recent survey of structural XML clustering algorithms covers 23 state-of-the-art approaches and highlights the main open issues in this domain [PBML14]. Another exhaustive survey covering the entire XML clustering process discusses both structure- and content-based methods [AMNS11]. Currently, the most cited survey related to XML clustering by structure [But04] concentrates on similarity computation. Other reviews limited to similarity computation include [GMS07], where eight measures are discussed, not all of which are XML-specific, and [TCY09] which concentrates on tree-edit distance measures also for schema comparison and content-based information retrieval. Finally, several reviews are limited to specific application fields. For example, [VPA07] and [HPRS07] focus on general clustering methods for Web documents.

Despite a tremendous effort put into the XML clustering research, the problems addressed in this thesis are still open. All of the described approaches are defined around local information and all except one require the number of clusters to be known a priori. Moreover, only a few methods tackle the issue of cluster interpretability. Finally, all of the above presented algorithms, except for Aggarwal et al. [ATW+07], follow the same clustering framework described in Section 2.3, while no such framework has been proposed for pattern-based approaches.

3.2 Classification

As mentioned earlier, XML classification has not been studied as intensively as XML clustering. However, several approaches have been proposed [NVK+09, VNK+10] and will be described in this section. Candillier et al. [CTT05] proposed a modification of their earlier described clustering algorithm based on the

EM algorithm. Importantly, the authors address the same issue as one of the issues stated in this thesis, i.e., producing easily interpretable results, by generating a decision tree in addition to the actual result. As a consequence, each cluster and class assignment can be summarized by a series of decisions. Garboni et al. [GMT05] proposed an approach where XML documents are represented as sequences and clustered in the training phase. For the classification phase, the authors propose several document-cluster similarity measures used to predict classes of new documents based on the cluster classes. Yang and Wang [YW09] proposed a method where XML documents are mined for closed frequent subtress, represented in a vector space model, and classified with SVM.

The most popular nearest neighbor approach (see Section 2.4) in XML classification was proposed by Bouchachia and Hassler [BH07]. In this approach, documents are represented as trees and classified based on k nearest training documents according to tree-edit distance. The authors analyzed both content and structure and arrived at a conclusion that structure is a more valuable source of information for XML classification purposes than content.

Regarding rule-based XML classifiers (see Section 2.4), an algorithm considered to be the state-of-the-art approach is XRules, proposed by Zaki and Aggarwal [ZA06]. The authors use an original mining algorithm to search for frequent embedded subtrees. These subtrees, along with their corresponding classes, form a set of rules, ranked according to their confidence, support, size, and lexicographical order. New documents are classified according to the first matching rule with the highest ranking.

Recently, a similar approach was proposed by Costa et al. [COR13] in an algorithm called X-Class. The authors also rely on an associative classifier, however, their proposal is generic and can incorporate any type of substructural features, such as element labels, paths, or subtrees. Moreover, unlike XRules, the training in X-Class does not produce rules with single-element antecedents. Instead, it treats each document as a transaction and finds common sets of frequent substructures. As a result, it is able to produce more powerful and discriminative rules, as evidenced by high quality results achieved in the experiments.

Each of the presented algorithms tackles some of the challenges stated in this dissertation, however, neither algorithm addresses them all. The closest approach is the one proposed by Zaki and Aggarwal [ZA06], however, it is prone to the default rule overuse problem as it concentrates solely on global information.

In addition to the above described structural approaches, several other XML classifiers have been proposed [TSW03, DG04, ZWB+11, LSG14]. However, they are designed for structure and content analysis and are, therefore, out of the scope of this thesis.

3.3 Approximate subtree matching

Tree-edit distance, along with many related problems, has been studied for many years now [Bil05, Tai79]. One of such related problems is approximate subtree matching, which is tackled in this thesis in the context of XML classification. One of the first attempts at solving this problem was proposed by Zhang and Shasha [ZS89]. The authors present a generalization of tree-edit distance, which can be stated as follows. Given trees t_1 and t_2, what is the minimum distance between t_1 and t_2 when zero or more subtrees can be removed from t_2 at no cost. This problem is similar to the question stated in this thesis, however, it works closely only when the root node of t_1 is mapped to the root node of t_2. Furthermore, it allows for all edit operations to appear in both trees. Given the XML classification motivation, the measure should be capable of identifying subtrees anywhere in the hierarchy of a tree and only by modifying a pattern tree in the least invasive way.

Tekli et al. [TCY07] proposed a new XML structural similarity measure based on tree-edit distance. In this measure, the authors introduce the notion of subtree commonalities to additionally account for the subtrees which appear multiple times in the compared trees. Although the authors also tackle the problem of subtree similarity, the overall goal and use of this information is different, as they focus on tree similarity while the measure presented in this thesis aims at measuring the degree of tree containment.

Another problem, similar to the one stated in this dissertation, was explored by Augsten et al. [ABBP10]. The problem concerned finding the best k matches of a small query tree in a large document tree. The authors propose a fast and scalable solution by focusing on efficient pruning. The algorithm prunes all excessive subtrees in a single post-order scan of a document tree, thus presenting linear complexity. This approach, however, is also unsuited for the stated problem because it focuses on subtrees spanning to the bottom of a document tree (leaf nodes) while for the purposes of XML classification the measure needs to identify subtrees of any shape and depth.

Recently, Cohen and Or [CO14] proposed a framework for solving the subtree similarity-search problem, along with an indexing structure to enhance the efficiency of the searching [Coh13]. Their solution is generic and allows for a wide variety of similarity measures to be used. However, the aim and scope of the framework is different to the problem addressed in this thesis. The authors focus on finding several similar subtrees using some subtree similarity measure while this work focuses on the sole problem of how to measure the subtree similarity. Therefore, the scopes of our work are different, nevertheless, complementary.

Much effort have also been put into XML tree patterns and tree pattern matching, which is a more general problem than the one stated in this thesis [HD13]. An interesting approach to tree pattern matching, called tree pattern relaxation, was proposed by Amer-Yahia et al. [ACS02]. The authors propose four relaxations of pattern constraints which allow for approximate pattern matching. This method

is dedicated for XML tree querying and, thanks to the applied relaxations, provides a ranking of results. However, it requires specifically constructed weighted patterns, while the patterns used in the classification algorithm proposed in this thesis are simple trees.

Another pattern matching related problem is approximate tree matching with variable length don't cares [ZSW94]. Zhang et al. adopted the idea of VLDC's from string matching to tree matching. Originally, a VLDC is a symbol represented by "*" which may appear anywhere in a string pattern and may substitute any substring of a data string. For example, given a pattern "te*is", the "*" symbol can substitute for "nn" in a word "tennis" or for "tr" in a word "tetris". Zhang et al. generalized this approach to tree matching by introducing two VLDC symbols in tree patterns: a path-VLDC "|" — a substitute for a path, and an umbrella-VLDC "∧" — a substitute for a path and all of its emerging subtrees. Finally, the authors expanded this approach to approximate tree matching by calculating tree-edit distance between pattern and document tree, given that substitution of a VLDC is performed at zero cost. This approach, similarly to tree pattern relaxation, requires patterns of a specific structure, which makes it unusable in this instance.

3.4 Conclusions

In this chapter, we have summarized the most relevant and characteristic algorithms in the discussed structural XML mining domains. The literature analysis concerned the two main fields of inquiry, i.e., XML clustering and XML classification, as well as the side issue of approximate subtree matching. All of the analyzed solutions tackle similar issues to the ones stated in this thesis, however, their detailed characteristics showcase that they do not resolve all of them.

Firstly, there is no generic methodology for clustering XML documents using global information while a local-information-based framework exists. Moreover, the issues of result interpretability and automatic number of clusters detection, which are of very high practical importance, are almost universally neglected. In the field of XML classification, either solely global- or local-information-oriented approaches have been proposed while the issue of default rule usage has not been addressed. An attempt to address this issue by exploring edit-distance-based subtree similarity measures reveals, that their detailed characteristics showcase that they are unsuited for this particular problem.

In the chapters to follow, we will systematically try to address these issues. We will begin by proposing a framework for clustering XML documents by patterns in the next chapter.

4

XPattern — a framework for clustering XML data by patterns

The previous chapter shows a tremendous effort put into developing new approaches in the XML clustering domain, evidenced by a large number of algorithms created in this field. In these approaches, various representations, similarity measures, and clustering algorithms have been explored and combined using the traditional framework described in Section 2.3. However, as highlighted before, this framework is based on direct document similarity and as such is focused primarily on using local information. Although it can incorporate global information, as shown for example by Aggarwal et al. [ATW+07], how such an incorporation should be accomplished is not defined within the framework and must be dealt with independently.

In this chapter, an alternative approach is proposed, focused on using global information expressed by patterns. It describes a pattern-based framework for clustering XML documents, called *XPattern*, which was presented in [PBM15]. First, an intuition and a conceptual description of the framework will be discussed in Section 4.1. In Section 4.2, the framework will be formalized. Finally, a generic algorithm will be presented in Section 4.3.

4.1 Conceptual description

The XPattern framework is based on a real-world observation that objects — for instance people — differ from each other in many details. Thus, we say that each person is unique. But if we omit the characteristic features, we can see that people manifest similar patterns of behavior that allow us to classify them into a limited amount of profiles (for example extroverts and introverts).

Following this analogy, we have to define what kind of patterns we are looking for, e.g., decide if we want to group people according to their behavior or maybe their appearance. Next, we have to identify all the patterns which appear in a given dataset. Since we do not know how many patterns we will find and there can

be even more patterns than people, we have to group the patterns into profiles. Once we have formed the desired number of profiles we can simply assign each person to the profile she/he fits best.

According to these observations, the XPattern framework is defined around four main steps:

1. Data transformation
2. Pattern mining
3. Pattern clustering
4. Document assignment

Let us now discuss each step of this framework in more detail.

Step 1 Data transformation

The purpose of the first step is to transform input data into a representation that allows for efficient pattern mining according to a chosen pattern definition. The framework is independent of any particular representation and thus, XML documents can be transformed into any of the data structures discussed in Section 2.1, such as trees or sets of tags.

Step 2 Pattern mining

In the pattern mining step, the transformed input dataset is mined for patterns. A pattern can be defined as any piece of information obtainable from a single document that has a user-specified property, e.g., appears frequently across the dataset. Typical structural patterns include frequent subtrees and tags, but features like tag counts, paths, or statistical measures can be used as well. As we are describing an abstract framework, we will not use any concrete pattern definition in the remainder of this section and discuss the consequences of particular definitions in Section 6.4.

A pattern definition implies a method which needs to be employed for pattern mining, e.g., patterns defined as frequent subtrees require a computationally expensive frequent subtree mining algorithm, while patterns defined as distinct tag counts require a simple dataset scan. Since in the discussed framework patterns serve as cluster representatives, the mining algorithm has to produce at least as many patterns as there are expected clusters. That is why, if there are less patterns than expected clusters, we have to restart the pattern mining step and search for new patterns.

Step 3 Pattern clustering

In the third step of XPattern, patterns are clustered into groups, called profiles. The framework does not define any pattern similarity measure necessary for clustering. Two basic approaches arise: i) similarity measures that compare pattern structures and ii) similarity measures that compare patterns by the number of the documents they co-occur in.

The first approach promotes structurally cohesive profiles and takes into account relations between patterns rather than between documents. The second approach promotes profiles with commonly co-occurring patterns and, thus, indirectly uses inter-document relationships. Additionally, the second approach requires less processing. Comparing pattern structures would require calculating similarity between each pair of patterns, while all information needed for clustering using document co-occurrences can be gathered during the pattern mining step. That is why, we propose to use the second approach.

Step 4 Document assignment

After patterns are grouped into profiles, in the final step of the framework we have to assign all documents to the clusters represented by these profiles. This is carried out by testing each document against each profile to check how well the patterns in that profile describe that particular document. A document is assigned to the cluster for which this test produces the best result. In this step, similarly as in the third step, we can use the earlier acquired information about the occurrences of patterns in documents. Therefore, we do not need to check whether a document contains a pattern. It is also worth noticing that some documents may not contain any pattern and, thus, may be left unassigned. This feature relates to the problem of outlier detection, as documents without corresponding patterns naturally form a set of outliers.

Let us now focus on the main differences between the steps of XPattern and the earlier mentioned typical clustering methodology [AMNS11]. Although, the first step of both methodologies transforms objects into a chosen representation, in the traditional approach this is done to compare documents with each other, while in our approach it facilitates the process of pattern mining. Pattern mining and pattern clustering are steps distinctive for our framework. In these steps we aim at tackling the challenge of creating easily interpretable cluster representatives in the form of profiles. The final step of our framework iteratively assigns all documents to profiles, incrementally creating clusters in the process. This differs from the traditional approach where the last step consists of clustering by direct document similarity, which uses only local information and usually cannot be performed incrementally. Finally, it is worth noticing that the XPattern framework facilitates outlier treatment, as it naturally captures documents which do not fit into any profile.

4.2 Formal definition

Let us now formalize the discussed framework. The first step aims at transforming all documents from a dataset $\mathcal{D}$ into a representation that permits the chosen type of feature extraction. Since data transformation is performed at the very beginning of the clustering process and all further operations concern the trans-

formed documents, for convenience and clarity of notation, we will use the same symbol $\mathcal{D}$ to denote a dataset both before and after the transformation.

Definition 1 *A **feature** f is any piece of information that can be extracted or calculated from an XML document, e.g., a subtree, element label, or total number of elements. We denote the set of all features by $\mathcal{F}$.*

Definition 2 *A document $d \in \mathcal{D}$ **contains** a feature f if f can be obtained from its transformed representation. We denote the containment relation by $f \in d$ and the number of occurrences of f in d by $m(f, d)$.*

Definition 3 *A feature f is called a **pattern** if it fulfills a user-specified predicate $pattern(f)$. We denote a single pattern by p and a set of all patterns by $\mathcal{P}$:*

$$\mathcal{P} = \{f \in \mathcal{F} : pattern(f)\}$$

The second step of the framework mines all available patterns $\mathcal{P}$ from the dataset, preserving the information about all document-pattern containment relations.

Definition 4 *A **set of profiles** Π^k is a set of k sets of patterns defined as follows:*

$$\Pi^k = \{\pi_1, \pi_2, ..., \pi_k : \forall_{i=1..k} \pi_i \neq \emptyset \wedge \pi_i \subseteq \mathcal{P} \wedge \forall_{j=1..k; j \neq i} \pi_i \cap \pi_j = \emptyset\}$$

Definition 5 *A **profile** π_i is an element of a set of profiles Π^k that represents a cluster c_i.*

In the third step of the XPattern framework patterns $\mathcal{P}$ are grouped into a set of k profiles Π^k. One can use any algorithm or similarity measure to create the profiles (see Sections 2.2 and 2.3) — they are only restricted by the definition of a profile and a profile set (Definitions 4 and 5).

Definition 6 *A document d is **connected** with a profile π_i if it contains at least one pattern from that profile:*

$$d \sim \pi_i \Leftrightarrow \exists_{p \in \pi_i} p \in d$$

Definition 7 *The degree in which a document and a profile are connected with each other (document-profile similarity) is measured by the **connection strength** function str, defined as follows:*

$$str : \mathcal{D} \times \Pi^k \to \mathbb{R}_0^+$$

Definition 8 *A document d is assigned to a cluster c_i if it has the highest connection strength with profile π_i:*

$$c_i = \{d \in \mathcal{D} : \arg\max_{\pi \in \Pi^k} str(d, \pi) = \pi_i\}$$

In the last step of the framework each document is assigned to a cluster according to Definition 8, and the process is complete. As a result, we obtain documents organized into k groups, each summarized with a separate profile.

4.3 Generic algorithm

Based on the presented definition of the XPattern framework, we can outline a simple, generic algorithm for clustering XML documents by patterns, which is presented in Algorithm 1. The algorithm takes a set of XML documents as an input and outputs a set of k clusters. Nominally, the number of clusters is a required parameter of the algorithm. However, some approaches may require additional parameters or no parameters at all, as will be shown in the next sections.

Algorithm 1 The XPattern generic clustering algorithm

Require: set of XML documents $\mathcal{D}$, number of clusters k
Ensure: set of k clusters $\mathcal{C}$
1: $\mathcal{D} \leftarrow TransformDocuments(\mathcal{D})$
2: $\mathcal{P} \leftarrow GetPatterns(\mathcal{D}, k)$;
3: $\Pi^k \leftarrow GroupPatterns(\mathcal{P}, k)$;
4: $\mathcal{C} \leftarrow k$ empty clusters each defined by one profile $\pi_c \in \Pi^k$;
5: **for all** $d \in \mathcal{D}$ **do**
6: $\quad c = \arg\max_{c \in \mathcal{C}} (str(d, \pi_c))$
7: $\quad$ add d to cluster c;
8: **end for**

The first step of the framework is the document transformation, carried out in line 1. Secondly, in line 2 the transformed documents are mined for patterns according to Definition 3. Thirdly, in line 3 the patterns are grouped together into profiles as defined in Definition 5. Finally, in lines 4–8 the documents are assigned to their respective profiles. Each document is tested against every profile for connection strength (Definition 7) and added to the respective cluster of the profile with the highest value (lines 6–7). After processing all of the documents in this manner, the algorithm terminates.

4.4 Conclusions

In this chapter, we have defined and motivated the XPattern framework for clustering XML documents by patterns. As it is a generic solution, there are parts which need to be further specified in order to create a complete, working algorithm. One can use any document representation, pattern definition, pattern similarity measure, and pattern clustering method to create an algorithm tailored to a specific problem.

In the next two chapters, two such algorithms will be described and experimentally evaluated. In Chapter 5 we will discuss a tree-based instance of XPattern while Chapter 6 will introduce a path-based approach.

5 XCleaner2 — a tree-based instance of the XPattern framework

This chapter presents the first instance of the XPattern framework. The algorithm is called *XCleaner2* and was presented in [BLMP11]. It is based on a tree representation with features defined as subtrees and patterns defined as maximal frequent subtrees. First, in Section 5.1 we will describe the proposed algorithm. Section 5.2 will illustrate how XCleaner2 operates with a simple example. Finally, in Section 5.3, we will empirically evaluate the proposed algorithm.

5.1 Algorithm

XCleaner2 uses a tree-based document representation proposed by Zaki [Zak02]. This representation relies on mapping the set of all XML tags in the dataset into integers uniquely identifying labels (label ids) and then encoding each document tree as a string. The encoding is carried out by adding label ids to the string in a depth-first pre-order traversal and adding a -1 symbol whenever backtracking from a child to its parent. For example, the tree in Figure 5.1 would be encoded as: 0 1 -1 2 2 2 -1 -1 -1 4 3 -1 3 -1 -1. The numbers in brackets next to tree nodes show the depth-first pre-order traversal.

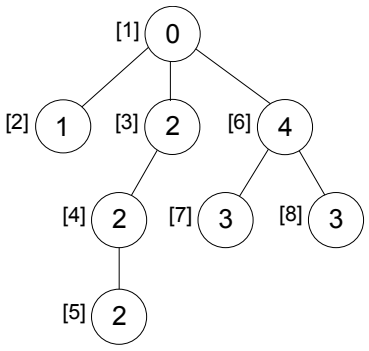


Figure 5.1: Tree representation of an XML document.

In XCleaner2, features are defined as subtrees. A feature f is called a pattern if it is a maximal frequent subtree. Let us recall the definition of frequency from Equation 2.1. A feature f is ***frequent*** if it is contained in at least $minsup$ percent of documents in the dataset $\mathcal{D}$, where $minsup$ is a user-defined minimum support parameter:

$$frequent(f, \mathcal{D}) \Leftrightarrow \exists_{\mathcal{D}' \subseteq \mathcal{D}} \forall_{d' \in \mathcal{D}'} f \in d' \wedge \frac{|\mathcal{D}'|}{|\mathcal{D}|} \geq minsup \tag{5.1}$$

A feature f is ***maximal*** if it is not contained in any other feature. In this case, if it is not a frequent subtree of any other frequent subtree found in the dataset.

$$pattern(f, \mathcal{D}) \Leftrightarrow frequent(f, \mathcal{D}) \wedge \neg \exists_{f' \in \mathcal{F}} \left(frequent(f', \mathcal{D}) \wedge f \subset f' \right) \tag{5.2}$$

We propose to use only maximal frequent subtrees so as to limit the number of patterns to a minimum. To find patterns defined as maximal frequent subtrees, *CMTreeMiner* algorithm proposed by Chi et al. [CXYM05] was used. CMTreeMiner discovers maximal frequent subtrees in a database of rooted, ordered, labeled trees by traversing an enumeration tree. An enumeration tree systematically enumerates all subtrees of an XML document. CMTreeMiner prunes the branches of the enumeration tree that do not correspond to maximal frequent subtrees, therefore rejecting groups of infrequent substructures rather than checking each substructure independently.

After obtaining the patterns, they are grouped according to their similarity. In this instance, we propose a pattern similarity measure based on document-pattern connections. The similarity between two patterns is directly proportional to the number of documents they share and is calculated according to the following formula:

$$sim(p_1, p_2) = \frac{|\{d \in \mathcal{D} : p_1 \in d \wedge p_2 \in d\}|}{|\{d \in \mathcal{D} : p_1 \in d \vee p_2 \in d\}|} \tag{5.3}$$

After calculating the similarity between all pairs of patterns, the results are stored in a similarity matrix and fed to the complete-link agglomerative hierarchical clustering algorithm (AHC, see Section 2.3) for clustering.

Once the profiles are obtained, we can start the document assignment process. Following the XPattern framework, we have to define how to measure the connection strength between documents and profiles (Definition 7). In XCleaner2, we propose to use the percentage of patterns in a profile contained in a given document:

$$str(d, \pi_i) = \frac{|\{p \in \pi_i : p \in d\}|}{|\pi_i|} \tag{5.4}$$

Connection strength is the final component required by the framework. Table 5.1 summarizes how each component of XPattern is defined in XCleaner2.

With all components defined, we can combine them into a complete, working solution. The pseudocode for XCleaner2 is listed in Algorithm 2.

The algorithm begins with encoding XML documents with the described string representation (line 1). In line 2, the algorithm mines for maximal frequent subtrees (see Equation 5.2) in the whole set of documents with support greater than

Table 5.1: XPattern components defined in XCleaner2.

XPattern component	XCleaner2 definition
Document representation	Tree (string-encoded)
Pattern definition	Maximal frequent subtrees
Pattern clustering algorithm	Complete-link AHC
Pattern similarity: $sim(p_1, p_2)$	$\frac{\lvert\{d \in \mathcal{D} : p_1 \in d \wedge p_2 \in d\}\rvert}{\lvert\{d \in \mathcal{D} : p_1 \in d \vee p_2 \in d\}\rvert}$
Connection strength: $str(d, \pi_i)$ (document-profile similarity)	$\frac{\lvert\{p \in \pi_i : p \in d\}\rvert}{\lvert\pi_i\rvert}$

Algorithm 2 XCleaner2 algorithm

Require: set of XML documents $\mathcal{D}$, number of clusters k, minimal support $minsup$
Ensure: set of k clusters $\mathcal{C}$

1: $Encode(\mathcal{D})$
2: $\mathcal{P} \leftarrow TreeMiner(\mathcal{D}, minsup)$;
3: $\Pi^k \leftarrow AHC(\mathcal{P}, k)$;
4: $\mathcal{C} \leftarrow k$ empty clusters each defined by one profile $\pi_c \in \Pi^k$;
5: **for all** $d \in \mathcal{D}$ **do**
6: $bestCluster \leftarrow$ the first cluster;
7: $bestMatchCount \leftarrow 0$;
8: **for all** $c \in \mathcal{C}$ **do**
9: $matchCount \leftarrow str(d, \pi_c)$; //Equation 5.4
10: **if** $matchCount > bestMatchCount$ **then**
11: $bestMatchCount \leftarrow matchCount$;
12: $bestCluster \leftarrow c$;
13: **end if**
14: **end for**
15: add d to $bestCluster$;
16: **end for**

or equal to *minsup*. As we can see, *minsup* is an additional parameter, absent in the generic algorithm. Afterwards, the patterns are compared with each other according to the measure defined in Equation 5.3 and clustered into k groups with a complete-link AHC algorithm (line 3). Finally, the documents are assigned to clusters based on the document-profile connection strength defined in Equation 5.4 (lines 4–16).

Before illustrating how XCleaner2 operates with a simple example, one more issue needs to be discussed. Namely, the CMTreeMiner algorithm used for pattern mining does not preserve the information about which patterns appear in which documents. In order to model this feature in our experiments, we proposed a subtree matching algorithm for string-encoded trees. However, as this algorithm is not an inherent part of the proposed clustering solution, for the clarity of presentation, it is described in Appendix A.

5.2 Example

Now, let us analyze a complete clustering example using the XCleaner2 algorithm. The clustering will be performed on 8 documents presented in Figure 2.2. We will cluster these documents into 2 groups and we will use $minsup = 0.5$ for pattern mining.

Step 1 Data transformation

In the first step, the documents are depth-first traversed and encoded as strings. The outcome of this process is illustrated in Tables 5.2 and 5.3. Table 5.2 presents the mapping of labels to label ids while Table 5.3 presents the string-encoded documents.

Table 5.2: Mapping of labels to label ids.

Label	Label id
paper	1
authors	2
person	3
publisher	4
inbook	5
title	6
year	7
volume	8
note	9
editors	10
journal	11

Table 5.3: String-encoded documents.

Document id	Representation
d_1	0 1 2 -1 -1 3 -1 4 5 -1 6 -1 7 -1 -1 -1
d_2	0 1 2 -1 -1 3 -1 4 5 -1 -1 8 -1 -1
d_3	0 1 2 -1 2 -1 -1 4 5 -1 9 -1 -1 -1
d_4	0 10 2 -1 2 -1 -1 4 5 -1 6 -1 7 -1 -1 -1
d_5	0 1 2 -1 -1 11 5 -1 6 -1 7 -1 -1 -1
d_6	0 1 2 -1 -1 11 5 -1 6 -1 -1 8 -1 -1
d_7	0 1 2 -1 2 -1 -1 11 5 -1 6 -1 7 -1 -1 12 -1 -1
d_8	0 1 2 -1 2 -1 -1 11 5 -1 6 -1 7 -1 -1 -1

Step 2 Pattern mining

In the second step, the CMTreeMiner algorithm is applied to obtain patterns. With $minsup = 0.5$, the algorithm finds 3 patterns presented along with their occurrences in documents in Table 5.4.

Table 5.4: Patterns and their occurrences in the documents.

Pattern id	Pattern	Documents
p_1	0 1 2 -1 -1 11 5 -1 6 -1 -1 -1	d_5, d_6, d_7, d_8
p_2	0 4 5 -1 -1 -1	d_1, d_2, d_3, d_4
p_3	7 -1	d_1, d_4, d_5, d_7, d_8

Step 3 Pattern clustering

In order to create profiles, we need to calculate the similarities between each pair of patterns. For example, patterns p_2 and p_3 together cover seven documents, two of which are common. Therefore, according to the formula in Equation 5.3, the similarity between p_2 and p_3 equals $2/7 \approx 0.29$. Performing analogous calculations for each pair results in a similarity matrix presented in Table 5.5.

Table 5.5: Pattern similarity matrix.

	p_1	p_2	p_3
p_1		0.00	0.50
p_2			0.29
p_3			

Once the similarity matrix is computed, the patterns are clustered into profiles using the complete-link AHC algorithm. Since the number of clusters k is 2, the AHC algorithm only needs one iteration in which two closest patterns will be grouped. According to the similarity matrix, patterns p_1 and p_3 are the most similar, so the profiles will have the following structure: $\pi_1 = \{p_2\}$ and $\pi_2 = \{p_1, p_3\}$.

Step 4 Document assignment

The final step assigns each document to one of the clusters based on the profile with which it has the highest connection strength. For example, document d_1 shares one pattern with profile π_1 and another one with profile π_2. However, as π_2 has two patterns while π_1 contains only one, the connection between d_1 and π_1 is stronger. According to Equation 5.4, the connection strength between d_1 and π_1 equals 1, while for π_2 it is only 0.5. Table 5.5 presents the connection strengths for all documents and profiles.

Table 5.6: The connection strength between documents and profiles.

	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8
π_1	**1.0**	**1.0**	**1.0**	**1.0**	0.0	0.0	0.0	0.0
π_2	0.5	0.0	0.0	0.5	**1.0**	**0.5**	**1.0**	**1.0**

According to Table 5.6, the documents will be clustered into two groups: one containing documents d_1 – d_4 and the second one containing documents d_5 – d_8. This concludes the algorithm and judging by the DTDs from Table 2.3, the outcome is correct.

5.3 Experimental evaluation

The proposed approach was experimentally evaluated on both real and synthetic data. XCleaner2 was compared with XProj [ATW+07], the tag-only approach [DA02] and the edge-only approach, where documents were represented as edge vectors (see Section 2.1). We also performed a sensitivity test w.r.t. *minsup* parameter. During the tests we used one real and three synthetic datasets. Unfortunately, we were unable to acquire the source code for XProj, so we decided to use the same datasets to make the comparison possible. We took the same real dataset, which is the XML SIGMOD database, and for generating the synthetic datasets we used the same software [BMKL02] and DTDs. For the evaluation of tag-only and edge-only approaches we used our own implementations with the complete-link AHC algorithm for clustering and the cosine distance for similarity evaluation (see Section 2.2).

5.3.1 Datasets and experimental setup

The XCleaner2 algorithm was implemented in the C# language and used a C++ implementation of the CMTreeMiner, which served for maximal frequent subtree mining. The experiments took place on a machine equipped with an Intel Pentium Dual Core E2140 @ 1,60 GHz processor and 3,00 GB of RAM.

The real dataset, denoted by sig, consists of 140 documents from the SIGMOD Record [SIG11]. The documents correspond to two DTDs: `IndexTermsPage.dtd` and `OrdinaryIssuePage.dtd` (70 XML documents for each DTD). The DTDs were naturally not used by the clustering algorithm, as this would render the task trivial.

Three synthetic datasets were used: one homogeneous and two heterogeneous. To generate these sets we used the ToXgene framework [BMKL02] and the same DTDs as Aggarwal et al. [ATW+07]. The homogeneous set, denoted by hom, contained 300 documents generated from 3 similar DTDs, 100 documents for each DTD. The *MaxRepeats* parameter, determining the maximum number of times a node will appear as a child of its parent node, was set to 3 for this dataset. The heterogeneous datasets, denoted by het3 and het6, both contained 1000 XML documents and were generated from 10 real DTDs, each of which was used to generate 100 documents. The *MaxRepeats* parameter was set to 3 for het3 and 6 for het6.

To evaluate the presented clustering method we used the Precision and Recall measures (see Section 2.5), defined as follows:

$$Precision = \frac{\sum_i s_i}{\sum_i s_i + \sum_i v_i}, \quad Recall = \frac{\sum_i s_i}{\sum_i s_i + \sum_i m_i}, \quad i = 1..k$$

where k is the number of clusters, s_i is the number of documents correctly assigned to cluster c_i, v_i the number of documents incorrectly assigned to c_i and m_i the number of documents which should be, but were not assigned to c_i. Each doc-

ument in every dataset has a label which indicates its corresponding DTD. Each cluster is identified by one, uniquely assigned DTD label. To assign a label to a cluster it should be the most frequent document label in that cluster and no other unassigned cluster should have more documents with that label. A document is correctly assigned to a cluster if they share the same label.

5.3.2 Parametrization

The selection of the *minsup* value is not an easy task because it highly depends on the dataset, in particular its homogeneity and distribution. However, if the number of clusters is given a priori (this is one of the assumptions of the algorithm) and the number of documents in clusters is suspected to be uniformly distributed, we suggest to set the *minsup* value to $\frac{1}{k}$, where k is the number of clusters. Before comparing XCleaner2 with other algorithms let us first analyze what impact does the *minsup* parameter have on the clustering quality.

To perform the *minsup* sensitivity analysis, we used the homogeneous dataset, as it is the most difficult one. Table 5.7 contains the results for *minsup* changing from 0.33 ($\frac{1}{k}$) to 0.8.

Table 5.7: Sensitivity test for *minsup* parameter (Precision, Recall and time).

minsup	Precision	Recall	Time [s]
0.33	**1.00**	**1.00**	0.91
0.40	0.62	0.62	0.70
0.50	0.67	0.67	0.33
0.60	0.67	0.67	0.16
0.70	0.37	0.37	0.09
0.80	0.42	0.42	**0.03**

The results clearly indicate that *minsup* has a big influence on both clustering time and quality of the results. When the parameter value rises, the clustering quality decreases but the algorithm runs faster. The quality drop is expected, because higher *minsup* means fewer and more general patterns, which in consequence leads to creating highly overlapping clusters. On the other hand, it also means that infrequent subtrees are cut more often, so the pattern mining time is significantly reduced.

5.3.3 Comparative analysis

XCleaner2 was compared with three other methods: XProj [ATW+07] (see Section 3.1) — the state-of-the-art algorithm and the tag-only [DA02] and edge-only approaches — two basic methods concentrated on efficiency. The two latter algorithms use the vector representation described in Section 2.1, the cosine distance measure described in Section 2.2, and the complete-link AHC algorithm described in Section 2.3. As explained in the previous section, for mining patterns in each dataset, we set *minsup* to $\frac{1}{k}$, where k is the desired number of clusters in a dataset. Table 5.8 presents the clustering results for each method.

Table 5.8: Precision and Recall for real and synthetic datasets.

Algorithm	*Tag-only*	*Edge-only*	*XProj*	*XCleaner2*
Dataset		Precision		
sig	**1.00**	**1.00**	**1.00**	**1.00**
het3	0.56	**1.00**	**1.00**	**1.00**
het6	0.52	**1.00**	**1.00**	**1.00**
hom	0.51	0.70	**1.00**	**1.00**
		Recall		
sig	**1.00**	**1.00**	-	**1.00**
het3	0.56	**1.00**	**1.00**	**1.00**
het6	0.52	**1.00**	**1.00**	**1.00**
hom	0.51	0.70	**1.00**	**1.00**

As the results show, all compared algorithms present the same quality in grouping data from the SIGMOD record. This illustrates the simplicity of this dataset, but also, more importantly, that some datasets simply do not require complex methods, as even an approach as basic as tag-only was able to produce perfect clusters. For the heterogeneous datasets, edge-only, XProj, and XCleaner2 still present high clustering quality, while the tag-only approach achieves results below 60%. These datasets were a bit more challenging, as the DTDs used to generate them had some common labels. As a result, the tag-only approach, based solely on these labels, was unable to detect the differences. However, adding only the information about parent-child relationships (edge-only) was sufficient to identify the groups flawlessly. Finally, the experiment involving homogeneous data illustrates that adding parent-child relationships may still be insufficient, as the edge-only approach achieved only 70% quality. This result confirms the intuition that data homogeneity highly influences the clustering outcome and the more homogeneous the data the more complex the methods are required.

The comparison between XProj and XCleaner2 is inconclusive, as both approaches present the same quality on all analyzed datasets. However, this result shows that the XPattern framework, which is a foundation for the XCleaner2 algorithm, is a valid alternative to the traditional framework which additionally systematically addresses the main problem of its rival, i.e., result interpretability.

5.4 Conclusions

In this chapter, we have discussed a tree-based instantiation of the XPattern framework, called XCleaner2. Thanks to using maximal frequent subtrees, the algorithm is capable of producing results of the highest quality even for more difficult datasets. The sensitivity tests showed that XCleaner2 is highly but predictably sensitive w.r.t. the minimum support parameter — the lower the value the better the result, in general.

Using complex patterns, such as frequent subtrees, allows to analyze documents with high precision, however, demands costly, time-consuming computations. As we will learn in the next chapter, such structures may be even too complex when large collections of wide documents are being processed. That is why, in the following chapter we will discuss another instance of the XPattern framework, capable of processing larger datasets. We will also examine the important issue of automatic approximation of the number of clusters.

6

PathXP — a path-based instance of the XPattern framework

This chapter presents the second instantiation of the XPattern framework — the *PathXP* algorithm — which was presented in [PBM15]. This approach is based on paths with patterns defined as maximal frequent subpaths. Section 6.1 describes the algorithm in detail while Section 6.2 illustrates it with a simple example. Section 6.3 discusses the problem of parametrization and describes methods for automatic number of clusters detection. Finally, in Section 6.4 the algorithm is experimentally evaluated.

6.1 Algorithm

In PathXP, a feature is defined as a subpath and a pattern is defined as a maximal frequent subpath (Equation 5.2). In accordance with the XPattern framework, the documents are first transformed into a chosen representation to facilitate the pattern mining process. PathXP relies on documents represented as multisets of full paths.

In order to discover frequent subpaths in the dataset, PathXP uses the Apriori algorithm [AS94] known from the market basket analysis. In this approach, documents serve as transactions and node labels serve as items. The only difference between our approach and the original is that when mining for frequent paths the order of items matters while in the classical algorithm it is irrelevant.

After obtaining the patterns, they are grouped together according to their similarity. In PathXP, as was the case in XCleaner2, pattern similarity is based on document-pattern connections. Therefore, the similarity between two patterns is directly proportional to the number of documents they share. Here however, the measure incorporates the information about the same pattern appearing several times in one document. The difference between these approaches will be discussed and experimentally evaluated in Section 6.4. Pattern similarity is calculated ac-

cording to the following formula:

$$sim(p_1, p_2) = \frac{\sum\limits_{d \in \mathcal{D}} \min\{m(p_1, d), m(p_2, d)\}}{\sum\limits_{d \in \mathcal{D}} m(p_1, d) + m(p_2, d) - \min\{m(p_1, d), m(p_2, d)\}}, \tag{6.1}$$

where $m(p, d)$ is the number of occurrences of pattern p in document d (Definition 2).

After calculating the similarity between all pairs of patterns, just like in XCleaner2, the results are stored in a similarity matrix and fed to the complete-link agglomerative hierarchical clustering algorithm (AHC, see Section 2.3) for profile creation.

Once the profiles are formed, the final phase of document assignment begins. Following the XPattern framework, a measure to calculate the connection strength between documents and profiles needs to be defined. In PathXP, connection strength is defined as the number of patterns contained in a document d which are present in a profile π_i, divided by the size of π_i:

$$str\,(d, \pi_i) = \frac{\sum\limits_{p \in \pi_i} m(p, d)}{|\pi_i|} \tag{6.2}$$

Dividing by the profile's size is used to eliminate the effect of promoting large profiles. It is also worth noting that unlike in XCleaner2, in PathXP the connection strength is not a normalized percentage of common patterns, as it can exceed 1 due to incorporating the information about patterns appearing several times in the same documents.

As an instance of XPattern, PathXP defines all of the components required by the framework, which are summarized in Table 6.1.

Table 6.1: XPattern components defined in PathXP.

XPattern component	PathXP definition
Document representation	Multiset of full paths
Pattern definition	Maximal frequent subpaths
Pattern clustering algorithm	Complete-link AHC
Pattern similarity: $sim(p_1, p_2)$	$\frac{\sum\limits_{d \in \mathcal{D}} \min\{m(p_1,d), m(p_2,d)\}}{\sum\limits_{d \in \mathcal{D}} m(p_1,d) + m(p_2,d) - \min\{m(p_1,d), m(p_2,d)\}}$
Connection strength: $str\,(d, \pi_i)$ (document-profile similarity)	$\frac{\sum\limits_{p \in \pi_i} m(p,d)}{\lvert\pi_i\rvert}$

The complete algorithm for PathXP is listed in Algorithm 3. It begins with setting the initial value of *minsup* to $1/k$ (the value $1/k$ will be discussed in detail in Section 6.3), where k is the number of expected clusters (line 1). Next, the input dataset $\mathcal{D}$ is mined for maximal frequent paths (line 2). Until the number of discovered paths is greater than or equal to the number of expected clusters, *minsup* is divided by 2 and the mining process is restarted (lines 3-6). This is

simply due to the fact that there need to be at least k patterns to form k clusters (at least one pattern per profile). The obtained set of patterns $\mathcal{P}$ is later grouped into k profiles using complete-link AHC algorithm with pattern similarity defined in Equation 6.1 (line 7). Finally, each document is assigned to a cluster with the highest document-profile connection strength defined in Equation 6.2 (lines 8–20).

Algorithm 3 The PathXP clustering algorithm

Require: set of XML documents $\mathcal{D}$, number of clusters k
Ensure: set of k clusters $\mathcal{C}$
1: $minsup \leftarrow \frac{1}{k}$;
2: $\mathcal{P} \leftarrow AprioriPaths(\mathcal{D}, minsup)$;
3: **while** $|\mathcal{P}| < k$ **do**
4: $minsup \leftarrow minsup/2$;
5: $\mathcal{P} \leftarrow AprioriPaths(\mathcal{D}, minsup)$;
6: **end while**
7: $\Pi^k \leftarrow AHC(\mathcal{P}, k)$;
8: $\mathcal{C} \leftarrow k$ empty clusters each defined by one profile $\pi_c \in \Pi^k$;
9: **for all** $d \in \mathcal{D}$ **do**
10: $bestCluster \leftarrow$ the first profile;
11: $bestMatchCount \leftarrow 0$;
12: **for all** $c \in \mathcal{C}$ **do**
13: $matchCount \leftarrow str(d, \pi_c)$; //Equation 6.2
14: **if** $matchCount > bestMatchCount$ **then**
15: $bestMatchCount \leftarrow matchCount$;
16: $bestCluster \leftarrow c$;
17: **end if**
18: **end for**
19: add d to $bestCluster$;
20: **end for**

Let us now analyze the worst-case complexity of the PathXP algorithm. The problem of mining frequent itemsets is known to be NP-Hard [Yan04]. The theoretical cost of mining frequent paths from a dataset of n documents with m distinct tags is $O(2^m n)$. The pattern clustering step uses AHC, hence for p patterns and k expected clusters the complexity of this step is $O(k \cdot p^2)$. Finally, the document assignment step searches for all occurrences of each pattern in each document. A single search for pattern occurrences in a document requires $O(v)$ operations, where v is the number of vertices in the document. Thus, the pessimistic complexity of the document assignment step is $O(p \cdot n \cdot max(v))$, where $max(v)$ is the number of vertices in the longest document in the dataset. Given the above, the overall worst-case complexity of PathXP equals $O(2^m n)$. Since the number of distinct labels m is usually bounded and n is the number of documents in a dataset, this shows that our algorithm can in practice scale linearly up to very large datasets (as evidenced by the scalability test performed in Section 6.4). Furthermore, since the most costly operation takes place in the pattern mining step, this solution should prove particularly effective in incremental clustering scenarios, where new documents are added to existing clusters.

6.2 Example

Just as with XCleaner2, let us now analyze a complete clustering example using the PathXP algorithm. Once again, we will cluster 8 documents presented in Figure 2.2 into two groups.

Step 1 Data transformation

First, the documents are transformed into multisets of full paths. The result of this transformation is illustrated in Figure 6.1.

$d_1 = \{$ paper/authors/person, paper/publisher, paper/inbook/title, paper/inbook/year, paper/inbook/volume$\}$	$d_2 = \{$ paper/authors/person, paper/publisher, paper/inbook/title, paper/note$\}$	$d_3 = \{$ paper/authors/person, paper/authors/person, paper/inbook/title, paper/inbook/number$\}$
$d_4 = \{$ paper/editors/person, paper/editors/person, paper/inbook/title, paper/inbook/year, paper/inbook/volume$\}$	$d_5 = \{$ paper/authors/person, paper/journal/title, paper/journal/year, paper/journal/volume$\}$	$d_6 = \{$ paper/authors/person, paper/journal/title, paper/journal/year, paper/note$\}$
$d_7 = \{$ paper/authors/person, paper/authors/person, paper/journal/title, paper/journal/year, paper/journal/volume, paper/pages$\}$	$d_8 = \{$ paper/authors/person, paper/authors/person, paper/journal/title, paper/journal/year, paper/journal/volume$\}$	

Figure 6.1: Documents represented as multisets of full paths.

Step 2 Pattern mining

After transforming the documents, the initial value of *minsup* is calculated. For the expected number of clusters $k = 2$, minimum support is set to 0.5. This means that for a dataset of 8 documents, a feature must occur in at least 4 documents to be considered frequent. Table 6.2 presents maximal frequent paths (patterns) found in the example dataset. The third column indicates the documents in which a given pattern occurs and how often. Since the number of obtained patterns is greater than the number of expected clusters, it is not necessary to decrease *minsup* and restart the frequent path mining process.

Table 6.2: Patterns and their occurrences in the documents.

Pattern id	Pattern	Documents
p_1	volume	d_1, d_4, d_5, d_7, d_8
p_2	paper/authors/person	d_1, d_2, $d_3(2)$, d_5, d_6, $d_7(2)$, $d_8(2)$
p_3	paper/inbook/title	d_1, d_2, d_3, d_4
p_4	paper/journal/title	d_5, d_6, d_7, d_8
p_5	paper/journal/year	d_5, d_6, d_7, d_8

Step 3 Pattern clustering

In the third step, the similarities between all patterns are computed and represented as a similarity matrix. Let us consider two example patterns p_2 and p_3. According to the formula in Equation 6.1:

$$sim(p_2, p_3) = \frac{1+1+1+0+0+0+0+0}{1+1+2+1+1+1+2+2} = 3/11 \approx 0.27.$$

Performing analogous calculations for each pair of patterns results in a similarity matrix presented in Table 6.3.

Table 6.3: Pattern similarity matrix.

	p_1	p_2	p_3	p_4	p_5
p_1		0.36	0.29	0.50	0.50
p_2			0.27	0.40	0.40
p_3				0.00	0.00
p_4					1.00
p_5					

After obtaining the similarity matrix, it is fed to the complete-link AHC algorithm for pattern clustering. Since the number of clusters k is 2, the algorithm will output two profiles: the first one (π_1) will contain pattern p_3 while the second one (π_2) — patterns p_1, p_2, p_4 and p_5. A dendrogram illustrating the clustering process is given in Figure 6.2.

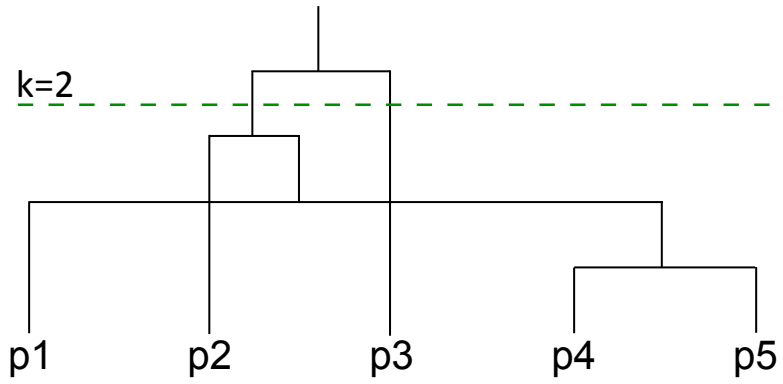


Figure 6.2: A dendrogram illustrating the process of pattern clustering.

Step 4 Document assignment

Finally, the documents are assigned to the profiles. Each document is assigned to the profile with which it has the highest connection strength (see Equation 6.2). As Table 6.4 shows, according to the number of pattern-document co-occurrences, documents d_1, d_2, d_3 and d_4 are assigned to the cluster represented by profile π_1, while documents d_5, d_6, d_7 and d_8 are assigned to the cluster represented by profile π_2. This concludes the algorithm.

Table 6.4: The connection strength between documents and profiles.

	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8
π_1	**1.00**	**1.00**	**1.00**	**1.00**	0.00	0.00	0.00	0.00
π_2	0.50	0.25	0.50	0.25	**1.00**	**0.75**	**1.25**	**1.25**

6.3 Parametrization

As stated in the introduction of this thesis, parametrization is an important issue, particularly in the clustering task. Having a possibility of manual parameter tuning may be desirable when a clear vision about the outcome is formed. However, as will be stressed later in this section, this usually is not the case with real-world clustering problems. Let us consider three possible scenarios of parametrization: i) k is given and *minsup* is unknown, ii) *minsup* is given and k is unknown, iii) both parameters are unknown. In this section, we will examine how this problem can be addressed in all of the above scenarios with the XPattern framework on the example of the PathXP algorithm.

Finding *minsup*

Choosing a proper minimum support threshold can be a problematic task. That is why, in PathXP we only require user to provide the number of clusters and we set the minimum support threshold automatically. Given the number of clusters k, similarly as in XCleaner2, we propose to set the minimum support parameter to $1/k$. The assumption behind this approach is that documents in each cluster share common patterns, therefore, assuming a uniform document distribution among clusters, the $1/k$ value should allow the algorithm for discovering these patterns. However, it is important to note that this approach may not be sufficient for highly imbalanced datasets.

Finding k

The requirement of knowing the number of clusters a priori is commonly assumed in most XML clustering algorithms. Recent XML clustering surveys [AMNS11, PBML14] reveal, that nearly all of the approaches proposed so far rely on this assumption. However, such a requirement may discredit the algorithm in many

real-world applications. To address this problem, we will propose two heuristics which automatically detect both *minsup* and the number of clusters. First however, let us consider a case when the number of clusters is unknown, but the correct value of *minsup* is given.

When the minimum support threshold is given, the number of clusters can be determined by analyzing the dendrogram obtained in the pattern clustering process. Given m patterns found in the dataset, the AHC algorithm requires $m-1$ steps to group them into one cluster. At each of these steps, two most similar clusters are merged. After creating the entire dendrogram, we can calculate the standard deviation σ of dissimilarities between the merged clusters. Analyzing all merge operations bottom-up, the first one that involves merging clusters based on a dissimilarity larger than 3σ is chosen as the cut-off point.

This approach is based on the 3σ rule [BMAD06], common in outlier detection. However, this rule needs to be slightly adapted to the conditions of our problem, because (as opposed to the outlier treatment problem) we have to find at least one cut-off point. Therefore, if no such point is found, we are iteratively altering the 3σ threshold using bisection, until obtaining at least one cut-off point, which produces the same value of k in two consecutive iterations.

Finding k and *minsup*

When neither k nor *minsup* are known, two approaches can be considered. The first approach interchangeably alters the values of both parameters using the σ threshold described above. The stopping condition is defined in terms of finding a certain number of patterns. First, we run the pattern mining algorithm with the *minsup* value equal 50%. If it does not find at least $|\mathcal{D}| \cdot 1\%$ patterns, where $|\mathcal{D}|$ is the number of documents, *minsup* is iteratively altered using bisection, until obtaining the required number of patterns, identical in two consecutive iterations. Secondly, the number of clusters k is determined, as described in the previous paragraph, by using the provided patterns. Lastly, if the previously processed *minsup* value is greater than $1/k$, we set its value to $1/k$, run the pattern mining step again, and process further with the pattern clustering into k clusters. However, if *minsup* was lower or equal $1/k$ we use the previously acquired pattern clusters at the processed cut-off point and proceed with the document assignment. The outline of this approach is given in Algorithm 4.

It is easy to notice, that the described heuristic relies on an artificial threshold of 1% which was selected experimentally. This solution seems to exchange one parameter (*minsup*) for another (number of patterns), however, defining how many patterns we would like to obtain may be easier than defining their minimal frequency in the dataset. Nevertheless, the proposed 1% threshold is, undoubtedly, dataset-dependent. To address this issue, we propose another variant, defined around a different stopping condition, namely, finding an empty cluster. This approach iterates over consecutive values of k, triggering PathXP, and converges after finding an empty cluster. Such a solution is dataset-independent and allows to detect the number of clusters in a methodical rather than parametrical manner.

Algorithm 4 Parameterless version of PathXP — quantity stop.

Require: set of XML documents $\mathcal{D}$
Ensure: set of clusters $\mathcal{C}$
1: $minsup \leftarrow \frac{1}{2}$;
2: $\mathcal{P} \leftarrow AprioriPaths(\mathcal{D}, minsup)$;
3: **if** $|\mathcal{P}| < |\mathcal{D}| \cdot 1\%$ **then**
4: $prevCount = -1$
5: **while** $|\mathcal{P}| < |\mathcal{D}| \cdot 1\%$ **or** $prevCount \neq |\mathcal{P}|$ **do**
6: $minsup$ bisection;
7: $prevCount = |\mathcal{P}|$;
8: $\mathcal{P} \leftarrow AprioriPaths(\mathcal{D}, minsup)$;
9: **end while**
10: **end if**
11: $dendrogram \leftarrow$ clustering of $\mathcal{P}$;
12: $\sigma \leftarrow$ standard deviation of distances between the merged clusters;
13: $threshold = 3\sigma$;
14: $k \leftarrow CutOffClusters(dendrogram, threshold)$;
15: **if** $k == 1$ **then**
16: $prevk = -1$
17: **while** $k == 1$ **or** $prevk \neq k$ **do**
18: $threshold$ bisection;
19: $prevk = k$;
20: $k \leftarrow CutOffClusters(dendrogram, threshold)$;
21: **end while**
22: **end if**
23: **if** $minsup > \frac{1}{k}$ **then**
24: $minsup \leftarrow \frac{1}{k}$;
25: $\mathcal{P} \leftarrow AprioriPaths(\mathcal{D}, minsup)$;
26: **end if**
27: $\mathcal{C} \leftarrow PathXP(\mathcal{D}, k)$; //Algorithm 3 without the first two lines

The pseudocode of this heuristic is illustrated in Algorithm 5.

This approach consists of two phases. In the first phase (lines 1-9), we determine the number of clusters by incrementally iterating through consecutive values of k and triggering our basic PathXP algorithm for each of these values. Once PathXP returns a result with at least one of the clusters empty, the iteration stops and we assume that the previous value of k was the correct number of clusters. After obtaining k, in the second phase (line 10) we simply use this information to cluster the dataset with PathXP.

This approach is based on an intuition that after reaching a certain number of clusters, profiles start to disintegrate and the connection strength between documents and these profiles weakens. This disintegration means, that the most unsuited patterns of the weakest profile form a separate profile. This profile should, eventually, get overwhelmed by other, more cohesive profiles, and be left with no documents assigned. This means that this profile is too weak to stand on its own and should be a part of another profile, thus, the number of clusters should be decreased.

Algorithm 5 Parameterless version of PathXP — empty cluster stop.

Require: set of XML documents $\mathcal{D}$
Ensure: set of clusters $\mathcal{C}$
1: $k \leftarrow 1$;
2: **while** $k < |\mathcal{D}|$ **do**
3: $\mathcal{C} \leftarrow PathXP(\mathcal{D}, k)$; //Algorithm 3
4: **if** $\mathcal{C}$ contains an empty cluster **then**
5: $k \leftarrow k - 1$;
6: **break**;
7: **end if**
8: $k \leftarrow k + 1$;
9: **end while**
10: $\mathcal{C} \leftarrow PathXP(\mathcal{D}, k)$; //Algorithm 3

The experiments show that the unparametrized versions of PathXP can produce results of similar quality to the parametrized version, however, they do not always accurately predict k with more difficult datasets. Details will be discussed in the next section.

6.4 Experimental evaluation

The proposed algorithm was evaluated in several experiments to inspect its properties and compare it with competitive approaches. In the following subsections, we will describe all of the used datasets, discuss experimental setup, and analyze the obtained results. In addition to the standard algorithm evaluation, we will also discuss alternative pattern definitions, test each major component of PathXP separately and analyze the number of clusters detectors.

6.4.1 Datasets and experimental setup

The proposed algorithm was tested against 5 real and 2 synthetic datasets which are summarized in Table 6.5. For real data, one heterogeneous dataset from the SIGMOD Record [SIG11] (sig) and four homogeneous datasets from the 2005/2006 INEX competition (db$X$, X = 0..3) were used. The db$X$ datasets were drawn from the IMDB movie database and ranked according to their difficulty — the higher the X, the more overlap there is between the classes. To generate synthetic datasets we used the ToXgene framework [BMKL02] with two sets of schemas: one containing 10 different DTDs for generating a heterogeneous dataset (het), second containing 3 similar DTDs for generating a homogeneous dataset (hom). The *MaxRepeats* parameter, indicating the maximal number of times that a node can appear as a child of its parent node, was set to 4 for all of the synthetic datasets.

All of the compared algorithms were implemented in the C# programming language. We also used a C++ implementation of the CMTreeMiner [Zak02]

Table 6.5: Characteristics of datasets.

Dataset	Classes	Number of documents	Avg size	Avg width	Avg height	Distinct labels
sig	2	140	82.66	32.16	5.46	39
db0-3	11	4825	220.03	112.24	5.32	195
het	10	1000	40.11	21.82	3.98	73
hom	3	300	36.37	18.07	4.00	12

algorithm for mining maximal frequent subtrees. The experiments took place on a computer with a 2,80 GHz Inter Core i7 processor and 16 GB of RAM.

To evaluate clustering quality we used Precision and a modified Recall measure (the standard Recall version produced the same result as Precision in all tests). The modified version reflects the dataset coverage and is defined as follows:

$$Precision = \frac{\sum_i s_i}{\sum_i s_i + \sum_i v_i}, \quad Recall = \frac{\sum_i s_i + \sum_i v_i}{|\mathcal{D}|}, \quad i = 1..k$$

where k is the number of clusters, $|\mathcal{D}|$ is the number of documents in the dataset, s_i is the number of documents correctly assigned to the i-th cluster and v_i is the number of documents incorrectly assigned to the i-th cluster. Although these measures originate from supervised learning, they can be used in our setting as all of the employed datasets contain labeled documents.

6.4.2 Alternative pattern definitions

Before discussing the properties of PathXP, let us analyze the impact of using various pattern definitions with the XPattern framework. As mentioned earlier, we propose to use XML paths, as they offer a compromise between full structural information and lightweight processing. This decision is supported by experimental results given in Table 6.6 which presents Precision, Recall, and clustering time of PathXP using subtrees, paths, tags, and metadata as patterns.

Subtrees are often presented as objects that best summarize the structural information of XML documents. They include all the information contained in tags or paths and provide additional sibling information. On the other hand, as experimental results show, subtrees are also the most expensive pattern definition from the compared set. Frequent subtrees are usually larger than tag or path patterns and, thus, more resource consuming. Furthermore, as stated by Chi et al. [CXYM05], the cost of mining maximal frequent subtrees is linearly proportional to the depth h of a document and exponentially proportional to its width w. This gives a worst-case complexity of $O(2^w)$ just for the pattern mining step. For this reason, although accurate for smaller datasets, this pattern definition cannot be used to cluster large, real-world datasets such as the INEX movie database. The experiments performed using subtrees as patterns on datasets db0-3 consumed all available memory before providing results, which is illustrated by blank values in Table 6.6.

Table 6.6: Precision, recall and clustering time for PathXP with different pattern definitions.

Dataset	sig	het	hom	db0	db1	db2	db3
Pattern				Precision			
Subtrees	**1.00**	**1.00**	0.90	-	-	-	-
Paths	**1.00**	**1.00**	**0.92**	0.66	**0.71**	**0.66**	**0.64**
Tags	0.51	**1.00**	0.35	**0.73**	0.69	0.45	0.44
Metadata	0.98	0.45	0.36	0.22	0.18	0.15	0.17
				Recall			
Subtrees	**1.00**	**1.00**	**1.00**	-	-	-	-
Paths	**1.00**	**1.00**	**1.00**	**1.00**	0.99	0.99	**1.00**
Tags	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Metadata	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
				Clustering time [s]			
Subtrees	0.06	1.12	2.88	-	-	-	-
Paths	0.15	1.44	0.09	65.25	137.76	311.64	403.38
Tags	0.08	1.06	0.05	21.44	25.62	42.66	44.07
Metadata	**0.02**	**0.03**	**0.01**	**0.18**	**0.24**	**0.40**	**0.38**

Simple sequences of tags are much easier to process than document trees. As time results presented in Table 6.6 show, tag patterns are among the easiest to acquire and process. This result finds its confirmation in the complexity analysis of tag mining, which in the worst-case scenario requires $O(M \cdot n)$ operations, where n is the number of documents in a dataset and M is the number of tags in the largest document in the dataset. However, precision acquired for the tag-based approach shows that tag patterns performed well only for the synthetically generated heterogeneous dataset (het) and for the noiseless real dataset (db0). This is consistent with the results obtained for the XCleaner2 algorithm. Such an outcome is expected as single tags can only characterize very distinct object groups. For more difficult homogeneous datasets (db1-3, hom), where most objects are very similar to each other, as well as for real heterogeneous dataset (sig), tags do not convey enough structural information to perform proper clustering. Additional data is needed for these datasets, such as additional structural information or metadata.

Alternatively, pattern definitions constructed from document structures can be replaced by data describing these structures, i.e., by structural metadata. In order to determine how well metadata can capture document characteristics, 10 different structural summaries were defined: number of elements, number of distinct elements, number of levels, average number of elements at all levels, standard deviation of number of elements at all levels, number of leafs, average path length, standard deviation of path length, average number of children for all nodes, and standard deviation of number of children for all nodes. For each dataset, a test involving all 1023 possible combinations of these parameters was carried out. The combination that performed best was based on two very basic structural

summaries: the number of distinct elements and the number of levels in a tree. The quality and time evaluation of this combination is presented in Table 6.6 (Metadata). The results clearly indicate that using metadata can produce clusters of competing quality compared to the tag-only approach for heterogeneous (sig, het) and synthetic homogeneous (hom) datasets, while requiring much less time. However, real homogeneous datasets (db0-3) reveal that this approach can only be used for simple problems or as additional information for other clustering methods.

Looking at the results a question arises: to what extent structural metadata can describe documents? For a closed set of XML data sources, documents can be clustered using solely structural metadata, but for very large datasets it seems more reasonable to use metadata only as additional clustering information. As denoted by Halevy et al. [HNP09], recent research shows that document processing requires the use of all available data and for this reason, the use of metadata such as document statistics can possibly become an important part of a real-world pattern definition, but as the presented results show, it is not sufficient to use them alone.

This thesis focuses mainly on clustering XML documents by structure, but it is worth noting that the presented approach can work equally well with textual data as patterns. One can use n-grams, synsets, or other word relations as cluster representatives. Since using patterns for clustering is a general idea, which does not imply their specific definition, it can be used with content as well as structure-based document representations. However, since XML clustering has many domains of application, we think that algorithms for this problem should give the user a choice on which information should be taken into account. For example, in domains such as chemistry, compounds can be encoded only by the structure of an XML format, thus eliminating the necessity for textual analysis. On the other hand, clustering records of a consistent, XML-based employee database can be performed without structural information. In other cases both structure and content should be taken into account.

6.4.3 Component analysis

When constructing the PathXP algorithm, several possibilities concerning algorithm functioning and parametrization were analyzed. The following tests present the results of this analysis. We will investigate how: restricting frequent patterns to maximal only, counting multiple pattern occurrences, and considering pattern uniqueness, influence the clustering quality. Additionally, we will analyze the impact of limiting the length of path patterns.

Let us start by analyzing three binary algorithm settings:

- **C**: Counting multiple occurrences of patterns in documents.
  This information is used by function $m(f, d)$ from Definition 2 to calculate pattern similarity and connection strength, as presented in Equations 6.1 and 6.2. In order to discard this information, we alter the definition of $m(f, d)$, so that it produces a binary result: 1 when $f \in d$, 0 otherwise.

- **W**: Weighting patterns according to their uniqueness.
  This information is expressed by the following formula:

$$WeightedSupport(p) = \sum_{d \in \mathcal{D}: p \in d} \frac{1}{|\{p' : p' \in d\}|},$$

  When used, it is embedded into the connection strength formula (Equation 6.2) as follows:

$$str\,(d, \pi_i) = \frac{\sum_{p \in \pi_i} m(p, d) \cdot WeightedSupport(p)}{|\pi_i|}$$

- **M**: Using only maximal frequent paths as patterns.
  This information is used in the $pattern(f)$ predicate in Equation 5.2. If we want to use all frequent paths, the $pattern(f)$ predicate simply changes to:

$$pattern(f) \Leftrightarrow frequent(f)$$

Table 6.7 presents the precision obtained by the algorithm for different combinations of settings {C, W, M}. Recall was omitted, as it remained unchanged for each dataset.

Table 6.7: Precision for varying algorithm settings.

Dataset			sig	het	hom	db0	db1	db2	db3	
C	*W*	*M*	Precision							Rank
0	0	0	**1.00**	**1.00**	**1.00**	0.62	0.54	0.51	0.47	4.9
0	0	1	**1.00**	**1.00**	**1.00**	0.54	0.51	0.54	0.49	5.0
0	1	0	**1.00**	**1.00**	0.76	0.49	0.28	0.17	0.18	6.7
0	1	1	**1.00**	**1.00**	0.76	0.61	0.54	0.53	0.50	5.4
1	0	0	**1.00**	**1.00**	0.92	**0.73**	0.68	0.58	0.55	3.1
1	0	1	**1.00**	**1.00**	0.92	0.66	**0.71**	**0.66**	**0.64**	**2.6**
1	1	0	0.50	**1.00**	0.51	0.72	0.63	0.55	0.52	4.9
1	1	1	**1.00**	**1.00**	0.79	0.65	0.70	0.65	0.62	3.4

In order to determine whether the presented settings significantly influence the quality of clustering, for every dataset we ranked each algorithm's performance from 1 to 8, where 1 is the highest and 8 is the lowest score. In cases when one or more algorithms were tied, average ranks were assigned (e.g., if two algorithms were tied at the 3rd place, each was granted a rank of 3.5). Once created, the ranking was used to perform the Friedman test [Dem06]. The null-hypothesis for this test is that there is no difference in the performance between the tested algorithm settings. Moreover, in case of rejecting this null-hypothesis we used the Bonferroni-Dunn post-hoc test [Dem06] to verify whether the performance of the best setting is statistically different from the remaining approaches. The result of this test is visualized in Figure 6.3.

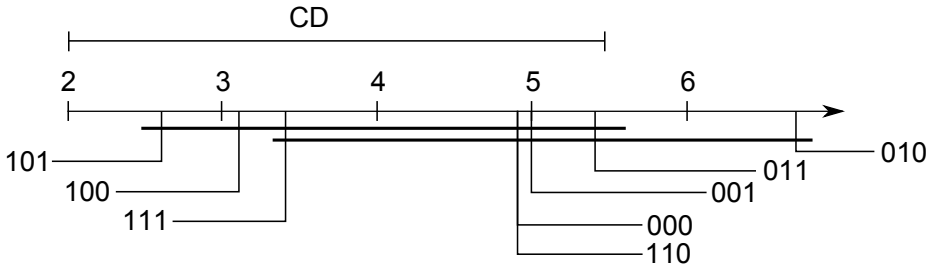


Figure 6.3: Friedmann test performed on the results from Table 6.7.

With 7 degrees of freedom, the value of the Friedman statistic equals 15.30, so with the significance level of $\alpha = 0.05$, we can reject the null hypothesis, what indicates that the algorithm settings are not identical. Furthermore, the Critical Difference (CD) chosen by the Bonferroni-Dunn test $CD = 3.5$ indicates, that settings 101 and 100 perform significantly better than setting 010. This means, that combining information about multiple pattern occurrences with maximal patterns is better than weighting patterns by their uniqueness combined with using all patterns instead of maximal. Additionally, analyzing the results of the two best settings (101 and 100), we observe that setting 101 performs equally good or better on all datasets, except for db0. That is why, in PathXP we use the combination of counting multiple pattern occurrences with maximal patterns.

Apart from analyzing different pattern counting schemes, we analyzed the possibility of limiting the maximal length of paths used as patterns. The results of this study are illustrated in Table 6.8.

Table 6.8: Precision for varying maximal path length.

Dataset	sig	het	hom	db0	db1	db2	db3	
Max path length				Precision				Rank
1 (tags)	0.51	**1.00**	0.35	**0.73**	0.69	0.45	0.44	3.6
2 (edges)	0.79	**1.00**	0.35	0.66	0.66	0.49	0.44	4.0
3	**1.00**	**1.00**	**0.96**	0.67	0.68	0.58	0.55	2.6
4	**1.00**	**1.00**	0.92	0.66	0.69	0.61	0.59	2.6
≥ 5	**1.00**	**1.00**	0.92	0.66	**0.71**	**0.66**	**0.64**	**2.1**

From the computational point of view, the most desired path length is 1 (tags), as longer paths yield exponential worst-case complexity in the pattern mining process. In order to determine whether path length changes the algorithm's quality we performed another Friedman test illustrated in Figure 6.4. This time we have five algorithms with path lengths ranging from 1 to 5 (the paths of length 5 and above produced the same results). For this setting, the Friedman test does not reveal a significant difference between the path lengths ($\alpha = 0.05$). However, because paths of unlimited length provided the best result for most of the datasets, in PathXP we are using frequent paths of unlimited length as patterns.

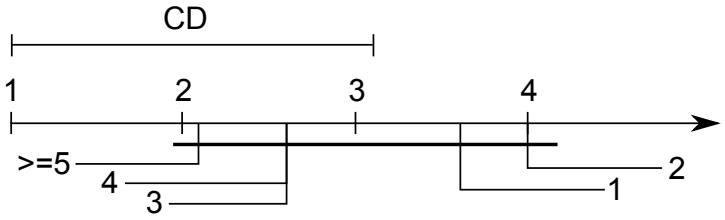


Figure 6.4: Friedmann test performed on the results from Table 6.8.

6.4.4 Parametrization

When designing the PathXP algorithm, a lot of effort was put into making it easy to parametrize. The following tests are designed to verify this claim. Additionally, a scalability test was performed to inspect how PathXP works with different dataset sizes. For this purpose, 20 heterogeneous datasets were generated, each containing from 5000 to 100000 documents. Each dataset was clustered with PathXP and time was measured for each step of the XPattern framework (except for Data transformation) separately. As results presented in Figure 6.5 show, the most time consuming stage of our algorithm is pattern mining, which is expected, as the worst case complexity of this step is $O(2^m n)$, where m is the number of distinct tags in the dataset of n documents. Additionally, the plot confirms that with the increasing number of documents, the execution time of our algorithm increases linearly.

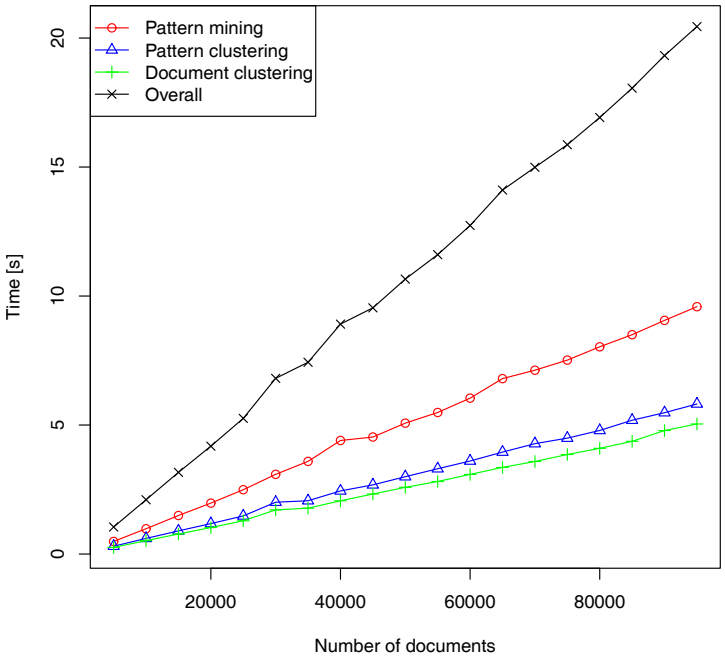
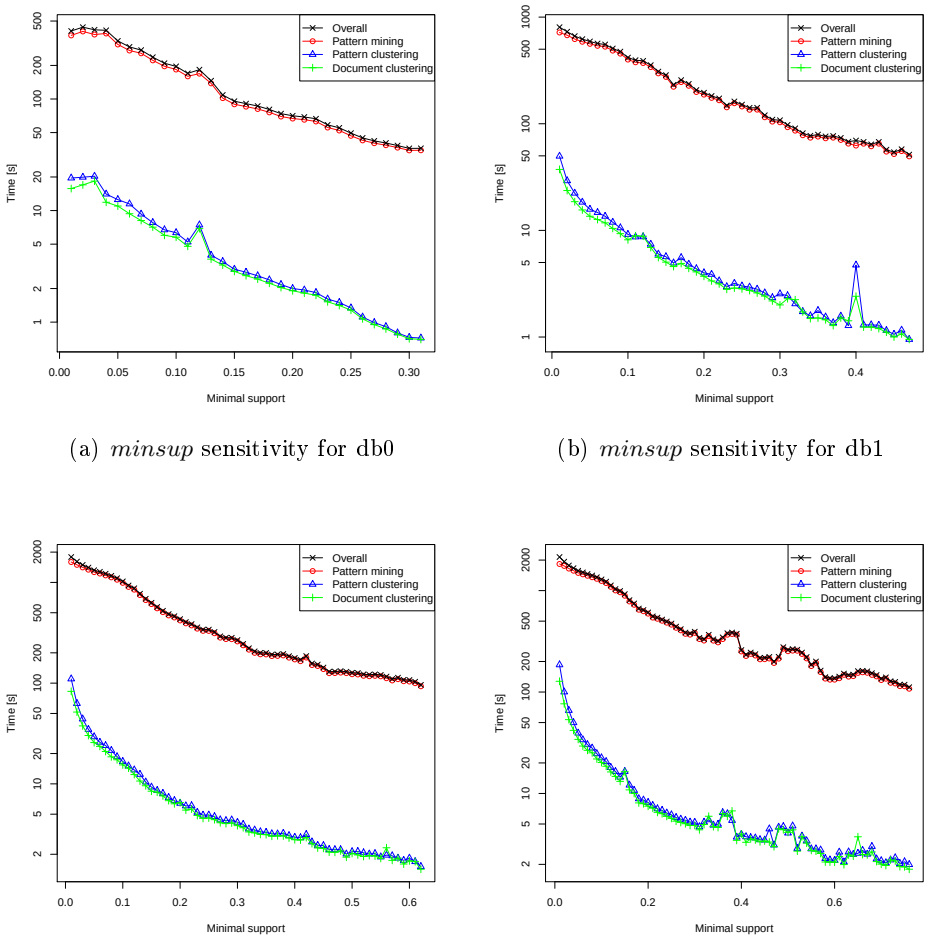


Figure 6.5: Scalability test results for PathXP

The parameter sensitivity analysis was performed using the db0-3 datasets, as they are the most difficult to cluster. As described earlier, the higher the dataset number the more noise there is in the data, so this should be reflected in this test. The *minsup* parameter was increased by 0.01 in every step, starting from 0.01

up until the pattern mining phase yielded too few patterns to form the required number of clusters. Figure 6.6 contains plots showing how minimum support changes the execution time of all steps in the algorithm.



(a) *minsup* sensitivity for db0

(b) *minsup* sensitivity for db1

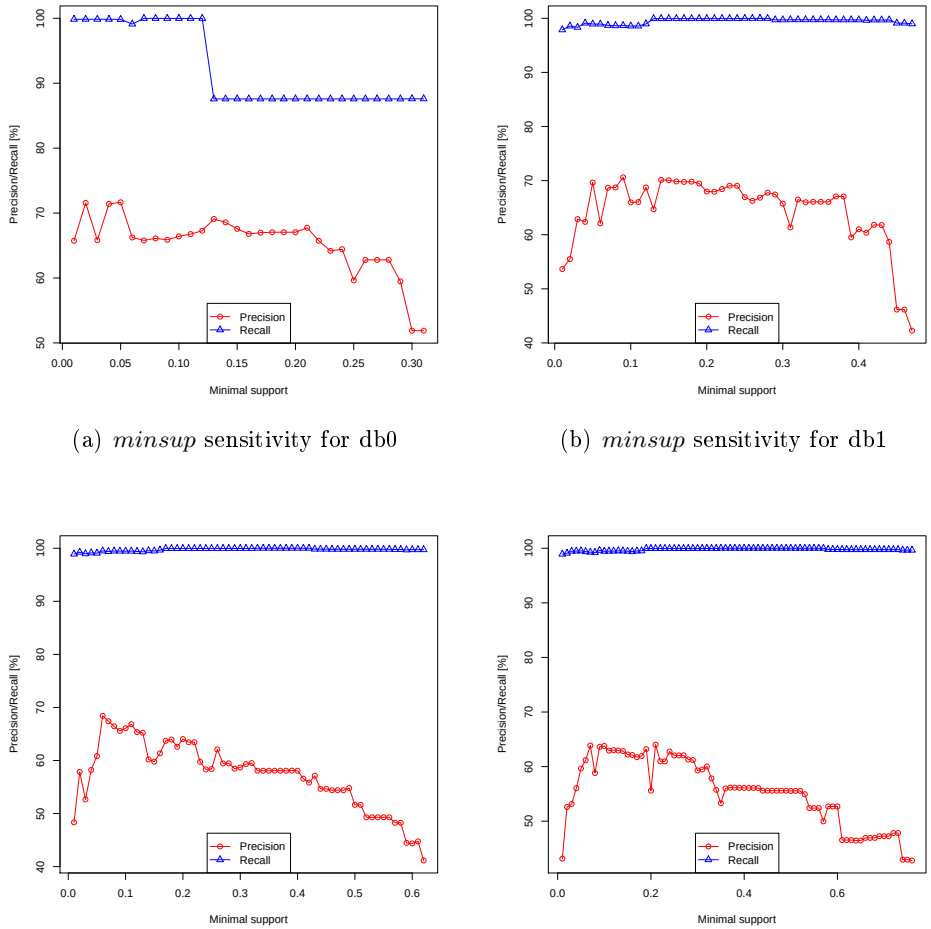
(c) *minsup* sensitivity for db2

(d) *minsup* sensitivity for db3

Figure 6.6: Sensitivity test results for PathXP w.r.t. minimal support: time

Similarly to the scalability test, the execution time depends mainly on pattern mining. The plots show, that with decreasing values of minimum support, execution time increases exponentially. This exponential growth also concerns the pattern clustering and document assignment. However, these steps do not use the minimum support parameter, therefore the shapes of the plots in fact illustrate the growth in the number of patterns generated during the pattern mining step. It is also worth noticing how marginal the cost of other steps is compared to pattern mining. Furthermore, as expected from the dataset characteristics, the more difficult the dataset, the longer it takes for the algorithm to terminate. Additionally, higher noise allows for higher *minsup* values to be achieved, as evidenced by the increasing lengths of the plots. This reflects the fact that noise spans across the whole dataset causing a lot of cluster overlap, so even though high *minsup* values are achievable, they produce patterns of weak discriminative power.

Let us now analyze how *minsup* influences the clustering quality. The outcome of this analysis is presented in Figure 6.7. As expected, clustering Precision decreases with the increase of the minimum support value. This is due to the fact that higher *minsup* values produce more general patters, which are not discriminative enough to distinguish between different clusters. However, it is worth noting that for a relatively wide range of *minsup* values (0.05 – 0.2), Precision holds a steadily high level in all tests. Only after the 0.2 threshold the quality starts to deteriorate notably in some datasets.



(a) *minsup* sensitivity for db0

(b) *minsup* sensitivity for db1

(c) *minsup* sensitivity for db2

(d) *minsup* sensitivity for db3

Figure 6.7: Sensitivity test results for the PathXP algorithm w.r.t. minimal support: Precision and Recall

The dataset coverage, reflected by the Recall measure, is much less affected by the parameter, for all datasets except for *db*0. In this case, it remains steady at approximately 100% for *minsup* between 0 and 0.13 and drops to 88% above the 0.13 threshold. This means that for *minsup* $>$ 0.13, approximately 12% of the documents have no corresponding patterns in any profile. In other words, after *minsup* reaches a certain threshold, the patterns generated in the mining process begin to reflect only the most common information in the dataset. As a result, in

the document assignment step, the documents with less common characteristics have no matching patterns and, therefore, are not assigned to any cluster. This, in turn, results in lower Recall. This property, however, is not present once the noise is introduced (datasets $db1$ – $db3$). As already mentioned, adding noise leads to more common information occurring across the whole dataset, so when mining for patterns, the algorithm is able to find them even for very high *minsup* values. However, these patterns have very weak discriminative power and lead to quality deterioration in terms of Precision.

The parameter sensitivity test (Figures 6.6 and 6.7) showcases that PathXP is predictably sensitive to the value of minimum support in terms of both execution time and clustering quality. The lower the parameter's value, the longer the execution time, but also higher clustering quality. It is worth noticing, that the suggestion of setting the value of this parameter to $1/k$ produces a compromise with satisfying quality and acceptable execution time.

Apart from the sensitivity test, we have also evaluated how our algorithm performs in the parameterless versions proposed in Section 6.3. Table 6.9 presents the number of clusters, Precision, Recall, and clustering time of PathXP and its two parameterless versions: PathXP-Q — where the stop condition is defined around finding a certain number of patterns (Algorithm 4) and PathXP-E — where the algorithm terminates once an empty cluster is found (Algorithm 5).

Table 6.9: Comparison of PathXP and parameterless PathXP.

Dataset	sig	het	hom	db0	db1	db2	db3
Algorithm	Number of clusters						
PathXP	2	10	3	11	11	11	11
PathXP-Q	2	10	2	8	23	23	23
PathXP-E	2	11	9	22	23	14	18
	Clustering time [s]						
PathXP	2	<1	<1	65	138	312	403
PathXP-Q	1	176	2	1617	2631	5546	4197
PathXP-E	1	19	1	1538	3176	3647	5999

The results in Table 6.9 show that the parameterless versions of PathXP were able to produce a reasonable approximation of the number of clusters. For heterogeneous datasets (sig, het) the results are identical for the quantity-based approach and nearly identical for PathXP-E — the algorithm found only one more cluster for het dataset than expected. For homogeneous datasets (hom, db0-db3), the algorithms behave differently. In the quantity-based approach (PathXP-Q) for noiseless homogeneous datasets (het, db1), the number of automatically detected clusters is below the expected value. For homogeneous datasets with noise (db2-db4), the PathXP-Q failed to detect the correct k and produced more clusters than expected. As a result, the clusters are smaller and more cohesive. In the empty-cluster-based approach (PathXP-E), the number of automatically detected clusters is higher than the expected value. The reason for this overestimation lies

in a fact that with homogeneous datasets there is naturally more overlap between the clusters than with heterogeneous datasets. This results in less cohesive profiles and leads to a more uniform distribution of connection strength, even for higher values of k. This, in turn, cases PathXP-Q to find empty clusters only after exceeding the actual number of clusters.

The additional cost of automatic cluster number detection is apparent in the processing time. For the analyzed datasets, the processing time of the parameterless versions is over an order of magnitude higher than the parametrized version. Such an overhead is a direct consequence of repetitive pattern mining in the case of PathXP-Q and triggering of PathXP in the case of PathXP-E.

6.4.5 Comparative analysis

After establishing the properties of PathXP, we have conducted an experiment comparing the proposed methods against 4 competitive structure based algorithms, which were described earlier in Section 2.3. This comparison was performed on datasets db$X$, as they were used by the authors of the competing methods. The results of this comparison are presented in Table 6.10 (Precision for algorithms other than PathXP was taken from the Report on the XML Mining Track at INEX 2005 and INEX 2006 [DG07]). The PathXP settings used for the comparison were default: counting multiple pattern occurrences with maximal, unlimited frequent paths as patterns.

Table 6.10: Comparison of PathXP with other structure based algorithms.

Dataset	db0	db1	db2	db3
Algorithm	Precision			
Vercoustre et al. [VFGL05]	0.45	**0.71**	**0.66**	0.53
Candillier et al. [CTT05]	0.78	-	-	-
Hagenbuchner et al. [HST+05]	**0.97**	-	-	-
Nayak and Xu [NI06]	0.60	0.60	0.59	0.59
PathXP	0.66	**0.71**	**0.66**	**0.64**

The comparison results are inconclusive, as they do not clearly indicate which approach is best. PathXP produced equally good or better results across all datasets compared to Vercoustre's et al., and Nayak's and Xu's approaches. On the other hand, Candillier's and Hagenbuchner's approaches outperformed PathXP on the first dataset (db0). Unfortunately, results for the remainig datasets (db1-3) were unavailable due to reasons not clearly stated by the authors [CTT05, HST+05]. Given the above, we cannot significantly state, that our algorithm performs better than the others. However, similarly as with XCleaner2, PathXP presents a competitive solution which additionally addresses the challenges stated in the introduction of this thesis.

6.5 Conclusions

In this chapter, we have discussed a path-based instantiation of the XPattern framework, called PathXP. By using maximal frequent paths as patterns, the algorithm produces easily interpretable, high quality results. Furthermore, we have examined the important issue of parametrization and discussed two approaches which automatically detect both the minimum support and the number of clusters. This topic is of particular importance in real-world applications, where the number of clusters is often unknown.

Examining the functioning of each component of the algorithm separately allowed us to draw some additional conclusions. By exploring different types of pattern definitions, from simple metadata to complex subtrees, we have discovered that frequent paths provide very good clustering quality while maintaining reasonable efficiency. We have also shown that the often omitted information about the number of occurrences of a pattern in a single document can significantly improve the clustering quality.

XCleaner2 and PathXP are based on similar, nevertheless significantly different assumptions. The use of frequent paths as patterns in the PathXP algorithm was dictated by a series of experiments conducted on several different definitions. While subtrees preserve more structural information than paths, the latter are easier to obtain and, thus, the whole clustering process is more efficient. That is why, we recommend using XCleaner2 for fairly small and highly homogeneous datasets and PathXP when larger collections are being analyzed.

PathXP concludes the three-chapter-long discussion on pattern-based XML clustering. In the next two chapters, we will build on the foundation of patterns and extend the idea to XML classification.

7 Partial tree-edit distance

The rule-based classifier, described in Section 2.4, is one of the most popular approaches in the XML document classification domain and serves as a foundation for the current state-of-the-art approach — XRules [ZA06]. However, as explained further in the same section, it suffers from the fact that if no matching pattern is found for a given document, it is assigned to one of the classes based on the default rule, which is somewhat arbitrary. To reduce this problem, one could use an approximate subtree matching measure to assign documents based on the closest, best-matching pattern. However, despite a substantial research done in the field of tree similarity computation and, as demonstrated in Section 3.3, in the field of approximate subtree matching, no measure is perfectly suited for this specific task.

In this chapter, we will discuss a measure designed specifically for the pattern-based XML classification task. The measure is called *partial tree-edit distance* (PTED) and answers the following question: Given two trees p (a pattern tree) and d (a document tree), find how much does p need to be modified to become a subtree of d using a constrained set of edit operations. These constraints concern both the set of operations and the tree nodes to which they can be applied.

The ability to efficiently determine trees whose subtrees are most similar to a given pattern tree p with respect to a constrained set of tree-edit operations is an important task not only in the context of XML classification, but also in problems such as XML querying or ranking [SM02, Yan04]. Furthermore, PTED can be used in the framework proposed by Cohen [CO14] (see Section 3.3), but in contrast to composite profile distance functions used in this framework, PTED returns the exact edit distance value between p and d.

Let us now describe partial tree-edit distance in detail. First, in Section 7.1 we will introduce necessary definitions and notation. In Section 7.2 we will present a conceptual description of PTED. In Section 7.3 we will formally define the proposed measure. Section 7.4 discusses an efficient algorithm for calculating the proposed measure. In Section 7.5 we will experimentally evaluate the algorithm and analyze its complexity.

7.1 Preliminaries

In order to formally define partial tree-edit distance, we will need some additional definitions and notation, but first, let us recall some basic concepts defined earlier in Chapter 2. A ***tree*** t is a connected graph with $|t|$ nodes and $|t|-1$ edges. We call a tree t ***rooted*** if all edges in t are directed away from one designated node, called a ***root*** node. We denote a tree t rooted at a node x by t_x and a root node of a tree t by r^t. Consequently, a root node has an indegree equal 0 while nodes with outdegree equal 0 are called ***leaf*** nodes. If two nodes x and y are connected with an edge and x is closer to the root node than y, then x is a ***parent*** of y and y is a ***child*** of x. Children of the same node x are called ***siblings*** and the number of all children of x is denoted by $|x|$. We also designate a special node λ, called ***empty node***.

A rooted tree t is ***ordered*** if there exists a total order among all nodes in t. In our approach, we order the nodes according to the depth-first pre-order traversal. The fact that node x appears in a tree before node y is expressed by $x < y$.

A tree t is ***labeled*** if every node in this tree $x \in t$ has a label assigned to it, symbolized by $l(x)$. For convenience, hereinafter, a rooted, ordered, labeled tree will be referred to as ***tree***.

An ordered set of trees is called a ***forest***. A forest F containing trees rooted at all children nodes of a node x is denoted by F_x. The rightmost tree of a forest F is denoted by $\vec{F}$. A forest F without a tree t is symbolized by $F - t$ and the number of nodes in all trees in F is symbolized by $|F|$.

A tree s whose nodes and edges form subsets of nodes and edges of another tree t is called a ***subtree*** of t. We denote that s is a subtree of t by $s \subseteq t$. A subtree $s \subseteq t$ whose nodes consist of some node in t along with all of its descendants will be referred to as a ***full subtree***.

Let us now define the edit operations which can be performed on tree nodes. In general, there are three basic edit operations: insertion, deletion, and relabeling. By ***inserting*** a node x into a tree t at a node y, x becomes a child of the parent of y, taking y's place in the sibling order, while y becomes a child of x. This operation is illustrated in Figure 7.1 with the right arrow. When ***deleting*** a node x from a tree t, all children of x become the children of the parent of x. Consequently, when x is a root node, the result is a forest F_x. This operation is illustrated in Figure 7.1 with the left arrow. ***Relabeling*** a node x to y means changing a label of x to $l(y)$.

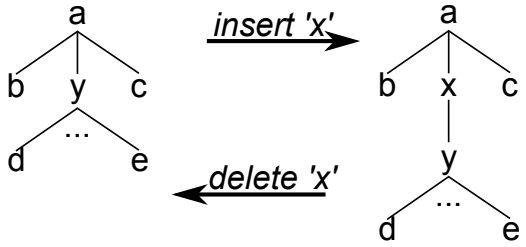


Figure 7.1: Node insertion and deletion.

7.2 Conceptual description

To illustrate how partial tree-edit distance works, let us consider the example presented in Figure 7.2. In this example, by t^i we will denote the i-th node (according to the depth-first pre-order traversal) in tree t. As the question stated by the measure implies, the task is to determine how many operations need to be performed on p for it to become a subtree of d. Looking at the example, clearly, p is not a subtree of d. However, as illustrated with the grey areas, there is a part of p which can be directly mapped into d. Namely, nodes p^2, p^4, and p^5 can be mapped into d^1, d^5, and d^{11}, respectively, as they have the same labels. As a result of this mapping, we also have to map p^3 into d^4. This time, however, we need to use the relabeling operation as the labels are different. Finally, as nodes p^1, p^6, and p^7 have no corresponding nodes in d, they have to be removed using the deletion operation. Therefore, the total number of edit operations required to transform p into a subtree of d is 4 (1 relabeling, 3 deletions).

So far, we have only used relabeling and deletion. Furthermore, we only deleted the root node (p^1) and the leaf nodes (p^6, p^7). Let us now discuss the possible consequences of using other edit operations, namely, deletion of inner nodes and insertion of inner and non-inner nodes. Inserting a non-inner node into p does not make sense, since it could only increase the number of operations needed to fit p into d. That is why, in partial tree-edit distance insertion of non-inner nodes is forbidden. As illustrated in Figure 7.1, deleting or inserting an inner node results in a children nodes' transfer, so the internal structure of a tree is altered. Inasmuch as this is permitted for embedded subtrees, it is forbidden for induced subtrees, which are the focus of the proposed measure.

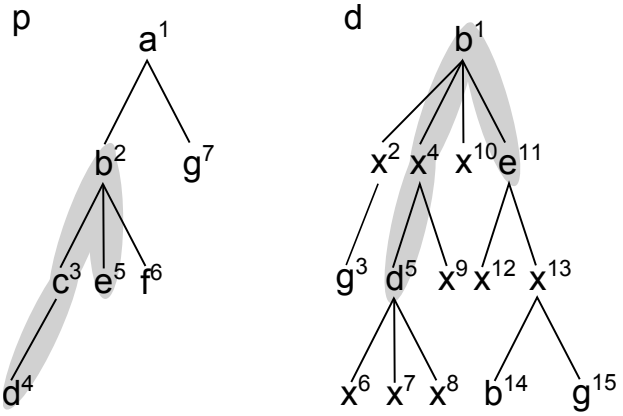


Figure 7.2: Example of fitting a pattern tree p into a document tree d. The nodes in p covered by the grey area are relabeled to the corresponding nodes covered by the grey area in d, while the nodes in p uncovered by the grey area are deleted. Numbers represent the order of depth-first pre-order traversal.

Given the above, partial tree-edit distance is defined around two edit operations: deletion of non-inner nodes and relabeling. Both of these operations have

an associated cost, which can be universally expressed with the following formula:

$$c(x,y) = \begin{cases} 0 & x = \lambda \\ w_d & y = \lambda \\ w_r & otherwise \end{cases} \tag{7.1}$$

where x and y are nodes, and w_d and w_r are user-defined weights associated with deletion and relabeling, respectively. Let s be a sequence of these two operations. ***Partial tree-edit sequence*** s between two trees p and d is a sequence which transforms p into any subtree of d. The cost $c(s)$ of partial tree-edit sequence s is the total cost of all operations in s. ***Partial tree-edit distance*** $\vec{\Delta}(p,d)$ between a pattern tree p and a document tree d is the minimal cost of all possible partial tree-edit sequences between p and d.

$$\vec{\Delta}(p,d) = min\{c(s) : s \text{ is a partial tree-edit sequence between } p \text{ and } d\} \tag{7.2}$$

7.3 Formal definition

Let us now formally define partial tree-edit distance. First, we will introduce the notion of partial mapping, which represents a partial tree-edit sequence.

Definition 9 *A **partial mapping** m between a pattern tree p and a document tree d is a subset of $p \times (d \cup \{\lambda\})$, such that:*

- *each node from p appears in m exactly once,*
- *each node from d appears in m at most once,*
- *for any $(x,x'),(y,y') \in m$ where $x' \neq \lambda$ and $y' \neq \lambda$: x is a parent of $y \Leftrightarrow x'$ is a parent of y',*
- *for any $(x,x'),(y,y') \in m$ where x is a sibling of y and x' is a sibling of y': $x < x' \Leftrightarrow y < y'$.*

Each element in the mapping $(x,x') \in m$ represents a single edit operation and has an associated cost $c(x,x')$, as defined in Equation 7.1. An element where $x' = \lambda$ represents a deletion while an element where $x' \neq \lambda$ represents a relabeling. The cost $c(m)$ of a partial mapping m is the sum of costs of all elements in m.

Definition 10 ***Partial tree-edit distance*** $\vec{\Delta}(p,d)$ *between a pattern tree p and a document tree d is the minimal cost of all possible partial mappings between p and d.*

$$\vec{\Delta}(p,d) = min\{c(m) : m \text{ is a partial mapping between } p \text{ and } d\} \tag{7.3}$$

Now, let us discuss a recursive formula which calculates partial tree-edit distance. The formula works in two stages. The purpose of the first stage, performed

by the main function $\vec{\Delta}$ and defined in Equation 7.4, is to place p at each possible position in d.

$$\vec{\Delta}(p,d) = \min_{x \in p, y \in d} \left\{ \vec{\delta}(\{t_x\},\{t_y\}) + \sum_{\{z \in p : z \notin t_x\}} c(z,\lambda) \right\} \tag{7.4}$$

Next, for each placement of p in d, the second stage takes place. The goal of the second stage, performed by an auxiliary function $\vec{\delta}$ and defined in Equation 7.5, is to check how well does p fit in d, at a given placement. The function accepts two forests G and H as parameters and recursively considers 3 cases: ignoring the rightmost tree of H, deleting the rightmost tree of G, and fitting the rightmost tree of G into the rightmost tree of H.

$$\vec{\delta}(G,H) = \min \begin{cases} \vec{\delta}\left(G, H - \vec{H}\right) \\ \vec{\delta}\left(G - \vec{G}, H\right) + \vec{\delta}\left(\{\vec{G}\},\emptyset\right) \\ \vec{\delta}\left(G - \vec{G}, H - \vec{H}\right) + \vec{\delta}\left(F_{r^{\vec{G}}}, F_{r^{\vec{H}}}\right) + c(r^{\vec{G}}, r^{\vec{H}}) \end{cases} \tag{7.5}$$

Equation 7.6 defines the boundary conditions of the auxiliary function $\vec{\delta}$. The first two cases reflect the fact that the cost of fitting an empty pattern into any tree always equals 0, while the third case reflects the fact that the cost of fitting any non-empty pattern into an empty tree equals the cost of removing the whole pattern.

$$\begin{aligned} &\vec{\delta}(\emptyset,\emptyset) = 0 \\ &\vec{\delta}(\emptyset,H) = 0 \\ &\vec{\delta}(G,\emptyset) = \vec{\delta}\left(G - \vec{G},\emptyset\right) + \vec{\delta}\left(F_{r^{\vec{G}}},\emptyset\right) + c(r^{\vec{G}},\lambda) \end{aligned} \tag{7.6}$$

Let us now present some basic properties of the formulas defined in Equations 7.4, 7.5, and 7.6.

Lemma 1 *If t_x is a subtree of t_y rooted at y, then $\vec{\delta}(\{t_x\},\{t_y\}) = 0$.*

Proof. Equation 7.5 defines 3 cases of fitting a forest $\{t_x\}$ into a forest $\{t_y\}$ and selects the minimal one. From the fact that t_x is a subtree of t_y rooted at y, follows that $l(x) = l(y)$, so the cost of relabeling x to y equals 0. Therefore, if we choose the third case in the first iteration: $\vec{\delta}(\{t_x\} - t_x, \{t_y\} - t_y) = 0$, $c(x,y) = 0$, and the recursion proceeds with $\vec{\delta}(F_x, F_y)$. In the next iteration, for a non-empty forest F_x the cost of a second case is always greater than zero, because it involves a node deletion. From the assumption that t_x is a subtree of t_y rooted at y we know that every tree in F_x has a corresponding tree in F_y, i.e., a subtree of t_y rooted at one of the children nodes of y. Thus, at each consecutive recursive step $\vec{\delta}(G,H)$ we have two options to consider:

1. $\vec{G}$ is a subtree of $\vec{H}$ rooted at $r^{\vec{H}}$ — in this case we choose the third option from Equation 7.5: $c(r^{\vec{G}}, r^{\vec{H}}) = 0$ and the recursion continues.
2. $\vec{G}$ is not a subtree of $\vec{H}$ rooted at $r^{\vec{H}}$ — in this case we choose the first option which adds no cost and the recursion continues.

Considering these two options in each recursive step we reach either the first or the second boundary condition in Equation 7.6, so the total cost will be 0. ■

Theorem 1 $p \subseteq d \Rightarrow \vec{\Delta}(p,d) = 0$.

Proof. The formula in Equation 7.4 fits each full subtree in p into each full subtree in d using the auxiliary function defined in Equation 7.5. A part of this process is fitting p (p is a full subtree of p) into all full subtrees of d. In these cases, the aggregated cost of removing nodes from p in Equation 7.4 ($\sum_{\{z \in p : z \notin p\}} c(z,\lambda)$) equals 0, as all nodes from p take part in the fitting. Therefore, in these cases the whole cost comes from fitting p into some full subtree $s \subseteq d$: $\vec{\delta}(\{p\},\{s\})$. From the assumption that $p \subseteq d$ follows that d contains a full subtree s_x of which p is a subtree rooted at x. From Lemma 1 we know that if p is a subtree of s_x rooted at x, then $\vec{\delta}(\{p\},\{s_x\}) = 0$. If $\vec{\delta}(\{p\},\{s_x\}) = 0$ and $\sum_{\{z \in p : z \notin p\}} c(z,\lambda) = 0$ then $\vec{\Delta}(p,d) = 0$. ■

Lemma 2 $\forall_{p,d}(w_r = w_d = 1) \Rightarrow \vec{\Delta}(p,d) \leq |p|$.

Proof. This lemma implies that the maximal value of PTED depends solely on the size of a pattern tree and does not depend on the document tree. In Equation 7.4 each full subtree in p is fitted into each full subtree in d. Just as in Theorem 1, consider a case of fitting p (p is a full subtree of p) into any subtree $s_x \subseteq d$. As earlier established, the aggregated cost of removing nodes from p in Equation 7.4 ($\sum_{\{z \in p : z \notin p\}} c(z,\lambda)$) equals zero, as all nodes from p take part in the fitting. As a result, the whole cost comes from the auxiliary function $\vec{\delta}(\{p\},\{s_x\})$. If in the first recursive step of Equation 7.5 we choose the second option, we get: $\vec{\delta}(\{p\}-p, s_x) + \vec{\delta}(\{p\},\lambda)$. With $\vec{\delta}(\{p\}-p, s_x)$ we reach the second boundary condition from Equation 7.6, so the cost equals 0. $\vec{\delta}(\{p\},\lambda)$ meets the third boundary condition from Equation 7.6 where all nodes in p are removed at a total cost of $|p|$ (assuming $w_r = w_d = 1$). Because this outcome is totally independent from s_x it is always achievable and, thus, forms an upper bound of $\vec{\delta}$ and, consequently, $\vec{\Delta}$. ■

Theorem 2 $\forall_{p,d}(w_r = w_d = 1) \Rightarrow 0 \leq \vec{\Delta}(p,d) \leq |p|$.

Proof. Follows from Theorem 1 and Lemma 2. ■

Lemma 3 *For any two trees t_x and t_y, $\vec{\delta}(\{t_x\},\{t_y\})$ calculates the minimal cost of all partial mappings m between t_x and t_y, such that $(x,y) \in m$.*

Proof. From the assumption that $(x,y) \in m$ we know that in the first recursion of $\vec{\delta}(\{t_x\},\{t_y\})$ we will not meet any of the boundary conditions defined in Equation 7.6. Furthermore, the first two cases from Equation 7.5 also violate the assumption that $(x,y) \in m$, so only the last case is applicable. As a result, in the first iteration we have a cost of relabeling x to y ($c(x,y)$), which reflects the assumption $(x,y) \in m$, and the recursion continues with $\vec{\delta}(F_x, F_y)$. From now on, in each recursive step we have three possibilities:

1. x does not have any children — in this case F_x is empty, so the cost equals 0, regardless of whether y has any children or not because there are no operations available to perform if we have no nodes in a pattern tree. This case covers the first two boundary conditions from Equation 7.6 and all nodes which will not appear in the mapping m, namely, all nodes from F_y.
2. x has children but y does not — in this case F_y is empty, so in order to fit F_x into an empty set we have to delete the whole forest F_x, because only an empty set is a subset of an empty set. Thus, the cost equals $|F_x| \times w_d$. This case covers the third boundary condition from Equation 7.6 and all nodes from F_x which will appear in the mapping m as a pair with an empty node λ, namely, all nodes from F_x.
3. Both x and y have children nodes — in this case both F_x and F_y are not empty, so we have 3 further possibilities to consider:
   a) We dismiss the rightmost tree $\vec{F_y}$ from forest F_y and try to match F_x with F_y without $\vec{F_y}$. If F_y has only one tree, then $F_y - \vec{F_y}$ produces an empty set and we reach the second boundary condition from Equation 7.6. This case covers the first possibility from Equation 7.5 and all nodes which will not appear in the mapping m, namely, all nodes from $\vec{F_y}$.
   b) We delete the rightmost tree $\vec{F_x}$ from forest F_x and try to match F_x without $\vec{F_x}$ with F_y. If F_x has only one tree then $F_x - \vec{F_x}$ produces an empty set and we reach the third boundary condition from Equation 7.6. This case covers the second possibility from Equation 7.5 and all nodes which will appear in the mapping m as a pair with an empty node λ, namely, all nodes from $\vec{F_x}$.
   c) We map the rightmost tree $\vec{F_x}$ from forest F_x into the rightmost tree $\vec{F_y}$ from forest F_y and try to match F_x without $\vec{F_x}$ with F_y without $\vec{F_y}$. If F_x has only one tree, then $F_x - \vec{F_x}$ produces an empty set and we reach the third boundary condition from Equation 7.6. If F_y has only one tree, then $F_y - \vec{F_y}$ produces an empty set and we reach the second boundary condition from Equation 7.6. If both F_x and F_y have only one tree each, then $F_x - \vec{F_x}$ and $F_y - \vec{F_y}$ produce empty sets and we reach the first boundary condition from Equation 7.6. This case covers the third possibility from Equation 7.5. Nodes $r^{\vec{F_x}}$ and $r^{\vec{F_x}}$ will appear in the mapping m as a pair.

These cases cover all possible partial mappings m between t_x and t_y restricted by $(x, y) \in m$. Since at each recursive step we choose the option with minimal cost, $\vec{\delta}(\{t_x\}, \{t_y\})$ finds the minimal partial mapping m between t_x and t_y restricted by $(x, y) \in m$. ■

Theorem 3 *The formula given in Equation 7.4 correctly calculates partial tree-edit distance according to Definition 10.*

Proof. For any two trees p and d, $\vec{\Delta}(p, d)$ represents the minimal cost of all partial mappings m between trees p and d. The only task of the formula given in

Equation 7.4 is to trigger the $\vec{\delta}$ function for each placement of p in d and select the one with the minimal cost. Since in Lemma 3 we have established that $\vec{\delta}$ calculates the minimal cost of all partial mappings m between two trees t_x and t_y with the restriction that $(x, y) \in m$, Equation 7.4 correctly calculates partial tree-edit distance according to Definition 10. ■

7.4 Dynamic algorithm

In this section we will discuss an algorithm which will calculate the partial tree-edit distance measure defined in the previous section. Similarly to the formal definition presented in Equations 7.4, 7.5, and 7.6, the algorithm consists of two main components: i) the main loop $\vec{\Delta}$ which places p at every possible position in d and ii) the auxiliary function $\vec{\delta}$ which checks the quality of each placement. The algorithm for the main loop is a trivial implementation of Equation 7.4, so we will skip the pseudocode for this step. A straightforward implementation of the auxiliary function from Equation 7.5 yields a very inefficient algorithm of exponential complexity. To resolve this issue, the auxiliary function is implemented with a dynamic programming algorithm, given in Algorithm 6.

Algorithm 6 Partial tree edit distance algorithm: $\vec{\delta}(t_x, t_y)$

Require: trees t_x and t_y,
Ensure: a minimal cost of a partial mapping m between t_x and t_y with restriction $(x, y) \in m$

```
1:  tab ← [|x| + 1, |y| + 1]
2:  for j = 0..|y| do
3:      tab[0, j] ← 0;
4:  end for
5:  for i = 1..|x| do
6:      tab[i, 0] ← tab[i − 1, 0] + (|t_{x_i}|) · w_d;
7:  end for
8:  for i = 1..|x| do
9:      for j = 1..|y| do
10:         tab[i, j] ← min{
                tab[i, j − 1],
                tab[i − 1, j] + (|t_{x_i}|) · w_d,
                tab[i − 1, j − 1] + δ⃗(t_{x_i}, t_{y_j})
            };
11:     end for
12: end for
13: return tab[|x|, |y|] + (l(x) = l(y) ? 0 : w_r);
```

The algorithm accepts two trees t_x and t_y as parameters and outputs the minimal cost of a partial mapping between t_x and t_y, given that x is mapped into y. Variable tab stores the intermediate results of mapping the children nodes of x into the children nodes of y, so it is an $\mathcal{R}^{|x|+1 \times |y|+1}$ matrix (line 1). In lines

2-4 the top row in the matrix is initialized to 0. This reflects the fact that the subtrees in the right tree can be removed without any cost (ignored). In practice, it fulfills the second boundary condition from Equation 7.6. In lines 5-7, the left column is initialized with the cumulative cost of deleting consecutive subtrees of x ($tab[i,0]$ = cost of removing $t_{x_1}..t_{x_i}$). These values fulfill the third boundary condition from Equation 7.6. Lines 8-12 contain the main loop of the auxiliary function. It scans through all children nodes of x and y and for each pair x_i, y_j stores a temporary result $tab[i,j]$ which holds the minimal cost of mapping $x_1..x_i$ into $y_1..y_j$. This cost is computed in line 10 as the minimum of 3 expressions, reflecting the 3 options in Equation 7.5:

- $tab[i, j-1]$ accounts for ignoring the rightmost subtree from the right tree;
- $tab[i-1, j] + (|t_{x_i}|) \cdot w_d$ accounts for removing the rightmost subtree from the left tree;
- $tab[i-1, j-1] + \vec{\delta}(t_{x_i}, t_{y_j})$ accounts for mapping the rightmost subtree of the left tree into the rightmost subtree of the right tree.

At the end of this procedure, $tab[|x|, |y|]$ holds the minimal cost of mapping the children of x into the children of y (with descendants). Finally, by adding the cost of mapping x into y in line 13, we obtain the total cost of the minimal partial mapping between t_x and t_y with x mapped into y. This concludes the algorithm.

Let us now analyze the complexity of the presented algorithm. It is easy to notice, that the algorithm for the auxiliary function is an adoption of the original algorithm for the Levenshtein distance between two sequences, which has a quadratic complexity. Here however, the auxiliary function is called within the main loop which is also quadratic in time, so the overall complexity is $O(n^4)$. However, it is worth noting that the auxiliary function runs only as deep as is the height of the smaller tree, so since the pattern tree is usually much smaller than the document tree, in practice, the algorithm should be more efficient than the complexity suggests.

7.5 Experimental evaluation

Partial tree-edit distance was proposed to lay a foundation for a new pattern-based XML classifier, which will be discussed in the next chapter. However, before incorporating PTED into a working algorithm, let us empirically evaluate the usefulness of the proposed measure in the context of rule-based XML classification (see Section 2.4). In the experiments, we used our own implementation of the rule-based classifier, which proceeds as follows. In the training phase, the training dataset is mined for maximal frequent subtrees separately for each class. Afterwards, the subtrees along with their corresponding classes form rules $r = t_r \rightarrow c_r$, where t_r is a maximal frequent subtree and c_r is a class. Next, all rules R are arranged into a ranking according to their confidence, support, and size. In the classification phase, each document is tested against each rule

in the descending ranking order for subtree matching, and assigned to the class indicated by the first matching rule.

As described in Section 2.4, the major drawback of the rule-based approach is the fact that when a document is being classified and there are no matching rules, it is assigned to one of the classes arbitrarily, most commonly to the majority class in the training data. The goal of this experiment is to illustrate the importance of this problem and show how partial tree-edit distance can be used to address it. To do so, we compared two approaches to dealing with this problem. In the first approach, we used the majority class method. In the second approach, we used partial tree-edit distance to assign each ambiguous document d to one of the classes according to the following formula:

$$class(d) = \arg\max_{c \in \mathcal{C}} \left(\sum_{r \in R_c} \left(1 - \frac{\vec{\Delta}(t_r, d)}{|t_r|} \right) \right)$$

where $\mathcal{C}$ is a set of classes and R_c is a set of rules with class c. Intuitively, this formula measures the similarity of d with all subtrees in each class and assigns it to the class with the highest cumulative similarity.

7.5.1 Datasets and experimental setup

In the experiments we used both synthetic and real datasets created by Zaki and Aggarwal [ZA06]. The synthetic datasets ds1-4, were generated by the aforementioned authors and are composed of a training and a testing set containing between 60000 and 100000 documents. The real datasets cs1-3, each consisting of around 8000 documents, contain web logs categorized into two classes (for a detailed description see [ZA06]). Since they were not divided into training and testing sets, we used each for both purposes and cross-validated them with each other. By csXY we will denote the cs$X$ set used for training and cs$Y$ for testing. This gives us a total of 10 tests: 4 on synthetic and 6 on real data. The minimal frequency of a subtree required to consider it a rule (*minsup*) was 0.1% for ds datasets and 1% for cs datasets.

All approaches were evaluated using the Accuracy measure, defined in Equation 2.6:

$$Accuracy_e = \sum_{c \in \mathcal{C}} \left(\frac{1}{|\mathcal{C}|} \cdot \frac{|\mathcal{D}_c^{test}|}{|\mathcal{D}_c|} \right)$$

where $\mathcal{D}_c^{test}$ is the set of documents correctly assigned to class c and $\mathcal{D}_c$ is the set of all documents from class c. Additionally, we used the Wilcoxon signed-ranks test [Wil45] to determine whether the proposed solution significantly improves the quality of classification.

7.5.2 Combining PTED with a rule-based classifier

Table 7.1 presents the results of the experiment. The first column ("Dataset") represents the datasets used in each test. The second and the third columns show the quality of the compared approaches on each dataset. "MC" shows the accuracy

of the majority class approach while "PTED" shows the accuracy of the partial tree-edit distance approach. The fourth column ("X") presents the percentage of documents from the test set which were unmatched by any rule from the classifier. It illustrates the gravity of the default rule problem, which motivated the creation of patrial tree-edit distance. In every test, this problem concerned around half or more documents (e.g., for test ds3 which contains 100000 test documents there were 73906 documents without any matching rule).

Table 7.1: The accuracy of compared classifiers.

Approach	*MC*	*PTED*			
Dataset	Accuracy [%]		X	Diff	Rank
ds1	50.03	**51.91**	56%	1.88	5
ds2	**51.43**	47.70	70%	-3.73	6
ds3	52.28	**58.18**	74%	5.90	9
ds4	42.90	**52.20**	63%	9.30	10
cs12	61.67	**62.06**	47%	0.39	3
cs21	58.66	**58.68**	49%	0.02	1
cs13	61.05	**62.05**	48%	1.00	4
cs31	58.08	**62.67**	50%	4.59	8
cs23	59.88	**59.94**	47%	0.06	2
cs32	58.45	**62.73**	50%	4.28	7

The results clearly indicate that by using partial tree-edit distance we were able to improve the classification quality in almost every test (except for ds2). This outcome is confirmed by the statistical test. Column "Diff" presents the difference in quality between the compared approaches for each test and column "Rank" shows the rank determined by the corresponding difference. In the Wilcoxon signed-ranks test, the W-value calculated based on the "Rank" column equals 6. The critical value of W for $N = 10$ datasets and $\alpha = 0.05$ is 8, so we can reject the null-hypothesis and state that the PTED approach performed significantly better than the majority class method.

7.6 Conclusions

In this chapter, we have discussed a measure dedicated for approximate subtree matching problem, called partial tree-edit distance. By combining the features of subtree matching and tree-edit distance, this measure describes to what extent one tree is included in another. We have also discussed an algorithm which calculates the proposed measure in polynomial time. Furthermore, we have analyzed the result of an experiment involving rule-based XML classification enhanced with partial tree-edit distance to illustrate the usefulness of the measure and highlight the gravity of the default rule problem. The results show that PTED can significantly improve the classification quality over the baseline rule-based approach.

Apart from XML classification, the measure discussed in this chapter opens several possibilities of future research. It could be used to improve the quality of approximate subtree matching, XML querying, ranking, or clustering. Furthermore, it would be interesting to explore the possible consequences of adding the information about multiple subtree occurrences to the measure, as the results involving clustering (see Section 6.4.3) suggest this information can hold high value in some applications.

In the next chapter, we will discuss a new pattern-based XML classifier. The algorithm is called k-nearest patterns and is defined around the partial tree-edit distance measure.

8

K-nearest patterns algorithm for pattern-based XML classification

In the previous chapter, we discussed a measure capable of calculating the degree of containment of one tree in another. Furthermore, we demonstrated how this measure could be incorporated into a rule-based XML classifier to enhance its predictive capabilities.

In this chapter, we will discuss a new XML classification algorithm based on the partial tree-edit distance measure, called *k-nearest patterns* (kNP). The algorithm attempts to combine the advantages of the rule-based classifier with the main features of the nearest neighbor approach while addressing the main issues of these methods (see Section 2.4). kNP classifies documents using both global and local information, by assigning documents to classes based on a weighted majority voting of their nearest patterns in the classification model. This goal is achieved in two phases. In the training phase, which will be described in Section 8.1, a classifier is created based on maximal frequent subtrees (patterns) found in the training data. In the classification phase, described in Section 8.2, the classifier is used to predict classes of new documents through a weighted majority voting of k nearest patterns. After discussing both of these phases, we will illustrate the whole process with a simple example in Section 8.3. In Section 8.4 we will empirically analyze a series of algorithm's variations, regarding: vote weighting, neighborhood definition, and inter-class duplicate pattern treatment, evaluate how the algorithm's parameters influence its performance, and compare kNP with the classical rule-based XML classifier and the state-of-the-art XRules [ZA06].

8.1 Training

In this section, we will discuss the training of a classifier in the k-nearest patterns algorithm. The goal of training is to create a classification model (classifier) $\mathcal{M}$. In kNP, classification model is a set of patterns defined as ordered pairs $p = (t, c)$, where t is a maximal frequent subtree and c is its corresponding class.

For convenience, we will also refer to a tree and a class of a given pattern p by t_p and c_p, respectively. A classifier is created in three main steps: mining for maximal frequent subtrees separately for each class (*Pattern mining*), combining the patterns into a single classification model (*Model creation*), and normalizing the attributes of the patterns across the whole model (*Attribute normalization*). Let us now analyze each of these steps in detail.

Pattern mining

First, the training dataset $\mathcal{D}$ is split into subsets — one subset $\mathcal{D}_c \subset \mathcal{D}$ per each class $c \in \mathcal{C}$. Next, each subset is separately mined for maximal frequent subtrees with *minsup* parameter set uniformly for each class. For this purpose, similarly as in the previous chapters, we used the CMTreeMiner algorithm proposed by Chi et al. [CXYM05]. As a result, we get a family $\mathcal{P}$ of $|\mathcal{C}|$ sets of maximal frequent subtrees $\mathcal{P}_c, c \in \mathcal{C}$. Each set $\mathcal{P}_c$, called ***class profile***, represents a single class and is defined as follows:

$$\mathcal{P}_c = \{t : t \text{ is a tree } \wedge pattern(t, \mathcal{D}_c)\},$$

where *pattern* is the predicate defined in Equation 5.2.

Model creation

In the next step, a classifier $\mathcal{M}$ is built, based on the profiles $\mathcal{P}$ obtained in the previous step. It is achieved by combining all maximal frequent subtrees into a single set of ordered pairs $\{(t, c) : \exists_{c \in \mathcal{C}} t \in \mathcal{P}_c\}$, so that each tree forms a 2-tuple with a class of documents from which it was derived. However, because the mining process takes place separately for each class, the subtrees may not be unique nor maximal across different classes, as they originate from different sets of documents. It may happen, that a tree in one class is a subtree of a tree in another class, thus, when combined into a single model, such patterns may lead to ambiguous class assignments. To address this problem we propose three different model construction strategies: *Leave all*, *Remove duplicates*, *Remove embedded*.

Leave all strategy assumes that any ambiguities resulting from co-occurring or non-maximal subtrees will be resolved in the classification phase. In this case, a classifier is simply constructed by joining all trees with their corresponding classes into a single, unordered set.

$$\mathcal{M} = \bigcup_{c \in \mathcal{C}} \{(t, c) : t \in \mathcal{P}_c\}$$

Remove duplicates strategy is more restrictive and does not allow two patterns with the same tree to appear in the model. Consequently, if a tree appears in multiple profiles it is completely excluded from the model. This strategy assures, that each tree in the model is unique.

$$\mathcal{M} = \bigcup_{c \in \mathcal{C}} \left\{(t, c) : t \in \mathcal{P}_c \wedge \neg \exists_{c' \in \mathcal{C}, c' \neq c} t \in \mathcal{P}_{c'}\right\}$$

Remove embedded strategy is the most restrictive, assuring that each tree in the model is maximal. If a tree is a subtree or a supertree of another tree from a different profile, it is excluded from the model.

$$\mathcal{M} = \bigcup_{c \in \mathcal{C}} \{(t, c) : t \in \mathcal{P}_c \wedge \forall_{c' \in \mathcal{C}, c' \neq c} \neg \exists_{t' \in \mathcal{P}_{c'}} (t \subseteq t' \vee t' \subseteq t)\}$$

The consequences of using either of the proposed strategies will be evaluated later in Section 8.4.2. It is important to note that choosing either the *Remove duplicates* or the *Remove embedded* strategy may lead to a situation where one of the classes has no patterns in the model or, in the extreme case, where there are no patterns at all (the model is empty). In such cases, where possibly a highly homogeneous dataset is being analyzed, we recommend using the *Leave all* strategy. However, during our experiments such a situation did not occur (see Section 8.4.2).

Attribute normalization

Apart from a tree and a class, every pattern in the model is described by a set of attributes, namely: size, support, and confidence. These values are calculated analogously as in the case of the classical rule-based classifier (see Section 2.4). ***Size*** of a pattern p is defined as the number of nodes in its tree: $size(p) = |t_p|$. ***Support*** of a pattern p in class c is defined as a frequency of its tree t_p among the training documents within class c:

$$supp(p, c) = \frac{|\{d \in \mathcal{D}_c : t_p \subseteq d\}|}{|\mathcal{D}_c|}$$

For convenience, we will use a short notation $supp(p) = supp(p, c_p)$ to calculate the support of pattern p in its corresponding class c_p. ***Confidence*** of a pattern p in class c is defined as:

$$conf(p, c) = \frac{supp(p, c)}{\sum_{c' \in \mathcal{C}} supp(p, c')}$$

As with support, we will use a short notation $conf(p) = conf(p, c_p)$ to calculate the confidence of pattern p in its corresponding class c_p.

These attributes will be used in the second phase for classification purposes (see Section 8.2). In order to assure a fair and equal treatment of all attributes, we normalize each attribute across the whole model using simple feature scaling:

$$attr'(p) = \frac{attr(p) - \min_{p' \in \mathcal{M}} attr(p')}{\max_{p' \in \mathcal{M}} attr(p') - \min_{p' \in \mathcal{M}} attr(p')},$$

where $attr$ (i.e., $size$, $supp$, $conf$) is an old attribute value and $attr'$ is a new, normalized attribute value. After normalization, each attribute has a value between 0 and 1, assuring an equal contribution of each attribute when vote weighting is applied.

8.2 Classification

After completing the training described in the previous section, we can proceed with the classification phase. The goal of the classification phase is to assign each new, unlabeled document d to one of the classes $c \in \mathcal{C}$ based on the model $\mathcal{M}$ created in the training phase. The class assignment takes place in several steps. First, the distances between d and every pattern p in the model are calculated. Afterwards, with a given neighborhood definition, we select a subset of the nearest patterns from $\mathcal{M}$ and aggregate the weights of all patterns in this subset for each class into a single class score. In other words, each pattern from the nearest neighborhood votes for its corresponding class. A single vote can have a weight equal to 1 or can range from 0 to 1, depending on the applied weighting strategy. After the votes are collected, finally, document d is assigned to the class with the highest score.

There are three main components that need to be specified in the described classification process: How to evaluate the similarity between documents and patterns (a problem of *Distance measure*)? How to define the boundaries of a neighborhood with a given distance measure (a problem of *Neighborhood definition*)? How to use pattern attributes in the voting process (a problem of *Vote weighting*)? Let us now look into these questions in detail.

Distance measure

In order to limit the nearest neighborhood, we need to define a distance measure. The problem is, that we have to compute distances between objects in two different spaces: documents and patterns. We know however, how these spaces are related, because pattern trees are derived directly from documents as their subtrees. Therefore, we need a measure that calculates how much does one tree (i.e., pattern tree) need to be modified to become a subtree of another tree (i.e., document tree). As typical methods used in tree processing, such as tree-edit distance or subtree matching, are not appropriate for this specific problem, we use the partial tree-edit distance measure (PTED), described in Chapter 7, designed specifically for such tasks. Let us recall that PTED calculates how much does one tree need to be modified to become a subtree of another tree.

To avoid promoting small patterns, we normalize each distance with the size of the pattern, so that it falls into a $< 0 - 1 >$ range. The complete formula for calculating distance between a pattern p and a document d is given as follows:

$$dist(p, d) = \frac{\vec{\Delta}(t_p, d)}{|t_p|}, \tag{8.1}$$

where $\vec{\Delta}(t_p, d)$ is the partial tree-edit distance between a pattern tree t_p and a document tree d.

Neighborhood definition

With a distance measure specified, next, we have to define the neighborhood of a document d in the pattern space. Let us discuss two possible cases of nearest neighborhood definitions: instance-based and distance-based.

The *instance-based* approach is analogous to that in kNN algorithm: nearest neighborhood of a document d is defined as the k nearest patterns in a model $\mathcal{M}$, according to the distances calculated with the formula in Equation 8.1. This approach requires user to provide the k parameter. Formally, instance-based nearest neighborhood of a document d in a model $\mathcal{M}$ is defined as follows:

$$\begin{aligned} kNP(d,\mathcal{M}) = \bigcup_{\mathcal{X}\subseteq\mathcal{M}} \{p \in \mathcal{X} : \\ |\mathcal{X}| = k \wedge \neg\exists_{\mathcal{Y}\subseteq\mathcal{M},\mathcal{Y}\neq\mathcal{X}}(|\mathcal{Y}| = k \wedge \max_{y\in\mathcal{Y}}(dist(y,d)) < \max_{x\in\mathcal{X}}(dist(x,d)))\} \end{aligned} \tag{8.2}$$

The *distance-based* approach defines neighborhood by pattern distances rather than a fixed number of patterns. In this scenario, all patterns within a specified range r form the nearest neighborhood of a given document d. As a consequence, each classified document may have a different number of patterns in its nearest neighborhood. This approach requires user to provide the r parameter. Formally, distance-based nearest neighborhood of a document d in a model $\mathcal{M}$ is defined as follows:

$$NP^r(d,\mathcal{M}) = \{p \in \mathcal{M} : dist(p,d) \leq r\} \tag{8.3}$$

When defining the nearest neighborhood with either of these approaches, a couple of additional issues need to be resolved. Considering the definition of the instance-based neighborhood given in Equation 8.2, we have to deal with situations in which several patterns are at the same distance from the analyzed document. As an example, consider 4 patterns x, y, z, w, a document d, and distances between each pattern and d equal to $0, 0.5, 0.5$, and 0.7, respectively. Given $k = 2$, it is ambiguous whether to include pattern y or z into the nearest neighborhood. In such cases, we propose to use one of the three following strategies: k^+NP, k^cNP, or k^rNP.

The k^+NP strategy is the one presented in Equation 8.2 and it resolves the problem of ambiguity by including all patterns which are at the same distance from d as the k-th pattern (in the distance order). Consequently, in this strategy the nearest neighborhood may contain more than k patterns.

In k^cNP strategy, if there are several patterns at the k-th position, we order them according to their confidence, support, and size. If two or more patterns have the same distance, confidence, support, and size, precedence between them is decided randomly. Once the ranking is created, we select only the top m patterns ($m \leq k$), so that the nearest neighborhood contains exactly k patterns.

The last strategy (k^rNP) is the simplest one and is based on the original, basic kNN algorithm. In this case, if there are several patterns at the k-th position, we select m out of these patterns at random ($m \leq k$), so that the nearest neighborhood contains exactly k patterns.

Choosing one of the presented strategies resolves the problem of the nearest neighborhood ambiguity in the instance-based model. On the other hand, when defining the nearest neighborhood with the distance-based model, given in Equation 8.3, we have to deal with a situation in which no patterns fall into the defined range r. In this case, we propose to use the $k^{+}NP$ strategy of the instance-based model with $k = 1$, as it simulates the range expansion to the distance of the closest patterns.

The consequences of using either of the presented approaches will be evaluated later in Section 8.4.2.

Vote weighting

After selecting a subset of nearest patterns NP according to one of the presented nearest neighborhood definitions, the next task is to check how well does the analyzed document fit into each of the class profiles. This is achieved by aggregating the weights of all patterns corresponding with a given class in the nearest neighborhood. As a result, each class has a score calculated with the following formula:

$$score(d, c) = \sum_{\{p \in NP : p_c = c\}} w(p, d) \tag{8.4}$$

where $w(p, d)$ is a weight associated with pattern p for document d. The pattern weighting can be carried out in several ways: all patterns weighted uniformly with $w(p, d) = 1$; patterns weighted according to their distances from the classified document $w(p, d) = (1 - dist(p, d))$; patterns weighted according to one of their attributes $w(p, d) = attr(p)$; patterns weighted with a product of their attributes and their distance from the document, e.g., $w(p, d) = conf(p) \cdot size(p) \cdot (1 - dist(p, d))$. The consequences of using either of the presented pattern weighting strategies will be evaluated later in Section 8.4.2.

With a class score function and a weighting strategy, a document d is assigned to the class with the highest score:

$$class(d) = \arg\max_{c \in \mathcal{C}} (score(d, c)) \tag{8.5}$$

This concludes the classification phase.

8.3 Example

Let us now illustrate how kNP works with a simple example. Consider the training set of documents presented in Table 2.2. Given that documents d_{1-4} represent a group of book chapters (class c_1) and documents d_{5-8} represent a group of journal papers (class c_2), the task is to classify an unlabeled document d_x, presented in Figure 8.1, into one of the two classes c_1, c_2.

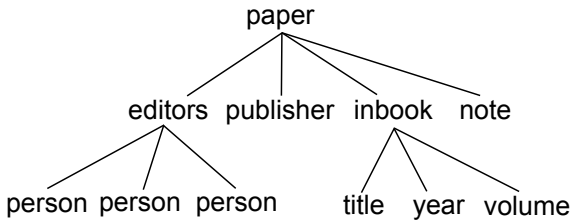


Figure 8.1: Example document d_x.

For this example, we will use a distance-based model ($r = 0.2$, $minsup = 0.5$) with *Leave all* strategy and pattern weights defined as $w(p, d) = (1 - dist(p, d)) \cdot conf(p)$.

In the training phase, first, we are mining for maximal frequent subtrees with a given $minsup$, separately for each class. The result of this process is illustrated in Fig 8.2. Next, since we are using the *Leave all* strategy, we create a model of the following structure: $\mathcal{M} = \{p_1 = (t_1, c_1), p_2 = (t_2, c_2)\}$. This concludes the training phase.

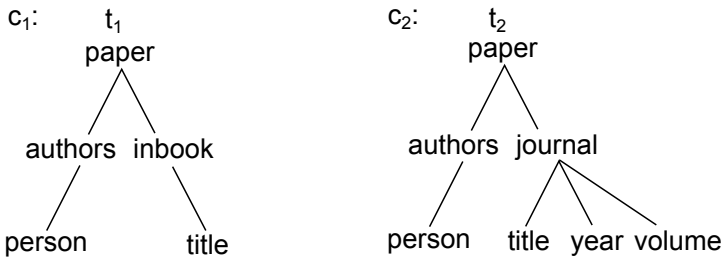


Figure 8.2: Patterns found in the example training dataset.

In the classification phase, we collect the nearest patterns NP of document d_x: $NP = NP^r(d_x, \mathcal{M})$. First, we have to calculate the distances from each pattern to d_x. Using the formula given in Equation 8.1, the distances for patterns p_1 and p_2 are 0.20 and 0.29, respectively. As we can see, pattern p_1 falls into the 0.2 range, so the nearest neighborhood has the following structure: $NP = \{p_1\}$. After selecting the nearest patterns, we calculate the score for each class. Since in this example the nearest neighborhood contains only one pattern the score computation is unnecessary. Nevertheless, let us perform the calculations for demonstrative purposes.

$$score(d_x, c_1) = (1 - dist(p_1, d_x)) \cdot conf(p_1) = (1 - 0.2) \cdot 1 = 0.8$$
$$score(d_x, c_2) = 0$$

Based on the calculated scores, document d_x will be assigned to class c_1, as it has the highest score. According to the DTDs from Table 2.3, this outcome is correct. This concludes the classification phase and the whole algorithm.

8.4 Experimental evaluation

The proposed approach was evaluated with a series of experiments, which will be discussed in this section. First, we will analyze how different components of the method, namely, model construction strategy, neighborhood definition, and vote weighting, influence the classification process. Afterwards, we will analyze the stability of the discussed approaches w.r.t. their parameters: *minsup*, k, and r. Finally, we will compare our approach with two other XML classification algorithms: the classical rule-based classifier and the state-of-the-art XRules algorithm [ZA06].

8.4.1 Datasets and experimental setup

To evaluate our method and fairly compare it with XRules [ZA06], we used the same datasets as Zaki and Aggarwal for the evaluation of their approach: 8 synthetic and 4 real. We have already mentioned these datasets earlier when demonstrating how partial tree-edit distance could be used to enhance the rule-based approach in the previous chapter. This time however, we used a slightly different real dataset configuration, consistent with the one used to evaluate XRules.

The synthetic datasets ds$X$.test and ds$X$.train ($X = 1..4$) were generated by the aforementioned authors. Each ds$X$.train dataset was used for training while its corresponding ds$X$.test dataset was used in the classification phase. The real datasets cs1, cs2, and cs3, contain web logs categorized into two classes (for a detailed description see [ZA06]). Additional dataset cs12 contains documents from both cs1 and cs2. Both training and testing datasets were labeled and this information was used for classification quality assessment. For convenience, in the following experiments we will use a shortened notation of the datasets: ds$X$ means an experiment performed with a pair of datasets ds$X$.train and ds$X$.test, while cs$X-Y$ means an experiment performed with dataset cs$X$ used for training and cs$Y$ for classification. All datasets are characterized in Table 8.1.

Unless stated otherwise, the parameters and components of our algorithms in the experiments were as follows: $k = 1$, $r = 0.2$, *Remove duplicates* model construction strategy, NP^r neighborhood definition, votes weighted with confidence and size, *minsup* as given in Table 8.1.

All algorithms were implemented in the C# programming language, with the exception of CMTreeMiner, implemented in C++ [CXYM05]. Additionally, the Accuracy of XRules was assessed from [ZA06]. The experiments took place on a machine equipped with a dual-core Intel i7-2640M CPU 2.8Ghz processor and 16 GB of RAM.

All approaches were evaluated using Accuracy, given in Equation 2.5:

$$Accuracy = \frac{|\mathcal{D}^{test}|}{|\mathcal{D}|},$$

where $\mathcal{D}^{test}$ is the set of correctly classified documents and $\mathcal{D}$ is the set of all docu-

Table 8.1: Datasets and their characteristics.

Dataset	Number of documents	Class distribution	*minsup*
ds1.test	88493	56.50%	-
ds2.test	72510	68.96%	-
ds3.test	100000	50.00%	-
ds4.test	74880	49.28%	-
ds1.train	91288	54.77%	0.008
ds2.train	67893	73.65%	0.009
ds3.train	100000	50.00%	0.007
ds4.train	75037	52.96%	0.002
cs1	8074	75.70%	0.035
cs2	7409	77.22%	0.045
cs3	7628	76.43%	0.045
cs12	15483	76.43%	0.040

ments. Additionally, to determine whether the analyzed components significantly influence the quality of our approach, in some experiments we used the Friedman test improved by Iman and Davenport, as proposed in [Dem06]. In such cases, for each dataset, each approach was granted a score from 1 to the number of tested approaches, where 1 is the highest score. The null-hypothesis for each test was that there is no difference in performance between all tested approaches.

8.4.2 Component analysis

Before comparing kNP with other approaches, let us analyze what impact does each of the components have on the classification quality. The goal of the first experiment was to select the best model construction strategy. Each approach, namely: *Leave all*, *Remove duplicates*, *Remove embedded*, was tested against all datasets. The result of this experiment is presented in Table 8.2.

Table 8.2: Model construction strategies.

Strategy	*Leave all*	*Remove duplicates*	*Remove embedded*
Dataset		Accuracy [%]	
ds1	73.05	**73.25**	72.00
ds2	**81.64**	**81.64**	65.35
ds3	65.86	**66.35**	65.10
ds4	**64.05**	**64.05**	62.17
cs1-2	80.33	80.33	**80.69**
cs2-3	**79.72**	**79.72**	79.05
cs3-1	**79.22**	**79.22**	78.90
cs12-3	79.40	79.40	**79.77**
Rank	1.875	**1.625**	2.500

The results show, that the *Remove duplicates* strategy performs best in 6 out

of 8 cases. Moreover, it produces equally good or better quality results than the *Leave all* strategy for all datsets. Interestingly, the *Remove embedded* strategy gives the worst accuracy on 6 out of 8 datasets, however, it performs best for cs1-2 and cs12-3 and competitively on the remaining two cs datasets.

In order to determine whether the discussed model construction strategies significantly influence the quality of kNP, let us analyze the results of the Friedman test. The average ranks for each of the strategies are presented in the last row of Table 8.2. With 3 algorithms and 8 datasets, F_F is distributed according to the F distribution with $3-1=2$ and $(3-1)(8-1)=14$ degrees of freedom. The F_F score for this experiment equals 1.784, so with $\alpha = 0.05$ and $F_{Critical} = 3.739$, we cannot reject the null-hypothesis. Thus, the test did not reveal any significant differences in quality between the analyzed approaches.

The results of this test are statistically inconclusive and do not clearly indicate the best solution. However, based on the above analysis, we propose to use the *Remove duplicates* strategy. Even though it was not significantly better than other strategies, it performed best on average and was the most stable approach across all datasets. Moreover, by choosing this strategy, we are able to reduce the size of the model without decreasing its predictive capabilities.

The aim of the second experiment was to compare the alternative neighborhood definitions. Each approach was tested against all datasets and the results of this experiment are shown in Table 8.3.

Table 8.3: Neighborhood definitions.

Neighborhood definition	k^rNP	k^cNP	k^+NP	NP^r
Dataset	Accuracy [%]			
ds1	65.26	71.25	72.62	**73.25**
ds2	77.06	74.20	80.61	**81.64**
ds3	62.24	65.57	65.70	**66.35**
ds4	60.56	63.94	63.28	**64.05**
ds1-2	79.59	49.37	**80.33**	**80.33**
ds2-3	78.96	79.68	**79.72**	**79.72**
ds3-1	78.76	**79.22**	**79.22**	**79.22**
ds12-3	78.66	79.35	**79.40**	**79.40**
Rank	3.750	3.000	1.938	**1.313**

The obtained results clearly indicate that the distance-based model (NP^r) produces the best results, as it outperformed or matched all other approaches in 8 out of 8 cases. The best instance-based model (k^+NP) produced equally good results for 4 datasets and the second best result in 3 cases. As expected, the definition which turned out to be the worst is the instance-based k^rNP model. It ranked at the last place in 5 out of 8 cases, outperforming only the k^cNP model in the remaining 3 tests.

Similarly as with the model creation strategies, let us perform a Friedman test in search for statistically significant differences. The average ranks for each of the

strategies are presented in the last row of Table 8.3. This time, we are comparing 4 algorithms on 8 datasets, so F_F is distributed according to the F distribution with $4-1=3$ and $(4-1)(8-1)=21$ degrees of freedom. Consequently, $F_F = 16.957$, so with $\alpha = 0.05$ and $F_{Critical} = 3.072$, we can reject the null-hypothesis and state that the analyzed neighborhood definitions produce significantly different results. Moreover, the Nemenyi test reveals that the distance-based model is significantly better than instance-based models k^rNP and k^cNP (critical distance $CD = 1.658$ for $\alpha = 0.05$). Comparing the instance-based models, k^+NP is also significantly better than k^rNP. Figure 8.3 illustrates the performed test and indicates the statistically significant differences.

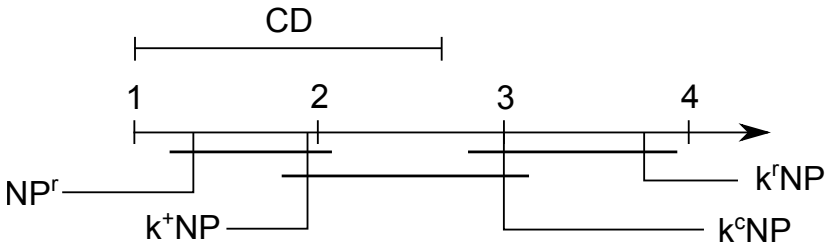


Figure 8.3: Friedman and Nemenyi tests for neighborhood definitions.

The results of this test clearly indicate that there is a substantial difference in classification quality between the analyzed neighborhood definitions. The distance-based model performs best, however, the instance-based k^+NP model produces results of competitive quality. As one would expect, the instance-based k^rNP model gives the worst results, as it resolves ambiguous situations in a random fashion. Based on the above analysis, we propose to use the distance-based model, as it performed significantly better than two of the instance-based approaches and matched or outperformed k^+NP model on all datasets.

The final component of the kNP classifier which needs to be discussed is vote weighting. Each weighting scheme was experimentally evaluated in order to check its impact on the classification quality. Since we have 4 possible weights, it gives us a total of 16 different combinations, all of which were tested on all datasets. The result of this experiment is presented in Table 8.4. Each row represents a different weighting strategy. The first four columns indicate if the corresponding weights were incorporated (1) or not (0).

The results reveal some interesting properties. Firstly, relying solely on the distance weight diminishes the accuracy when compared with the strategy without any weights in 4 out of 8 tests. Other than that, adding the information about distance seems to have a minor, if any, effect on the classification quality. Given the above, let us focus on the other weights without the distance incorporated ($Dist = 0$). The combination of confidence and size gives the best results on 4 out of 8 datasets (ds1, ds3, cs1-2, cs12-3) and ranks at a second place on the other 4. Additionally, analyzing the results vertically we can see that some datasets are much less affected by the changes in weighting strategy than others (especially cs2-3 and cs3-1).

The Friedman test confirms that altering the weighting strategy leads to sig-

Table 8.4: Weighting strategies.

Dataset				ds1	ds2	ds3	ds4	cs1-2	cs2-3	cs3-1	cs12-3	
Dist	*Conf*	*Supp*	*Size*	Accuracy [%]								Rank
0	0	0	0	72.76	80.89	65.26	61.48	77.66	79.72	**79.23**	76.49	5.625
1	0	0	0	72.44	80.58	64.98	61.45	77.66	79.72	**79.23**	76.49	-
0	1	0	0	73.00	81.61	66.18	64.45	80.31	79.68	79.22	79.37	3.750
1	1	0	0	73.00	81.61	66.18	**64.46**	80.31	79.68	79.22	79.37	-
0	0	1	0	70.55	79.85	63.97	62.99	79.35	79.72	**79.23**	78.80	5.625
1	0	1	0	70.37	79.85	63.96	62.76	79.35	79.72	**79.23**	78.80	-
0	1	1	0	72.46	81.50	66.02	63.33	80.31	79.69	79.22	**79.40**	4.375
1	1	1	0	72.46	81.51	66.02	63.29	80.31	79.69	79.22	**79.40**	-
0	0	0	1	73.04	81.02	65.85	62.83	76.10	**79.76**	79.22	73.81	5.250
1	0	0	1	73.04	80.77	65.85	62.84	76.10	**79.76**	79.22	73.81	-
0	1	0	1	**73.25**	81.64	**66.35**	64.04	**80.33**	79.72	79.22	**79.40**	**2.375**
1	1	0	1	**73.25**	81.64	**66.35**	64.05	**80.33**	79.72	79.22	**79.40**	-
0	0	1	1	71.10	80.61	65.31	62.92	77.76	**79.76**	79.22	76.34	5.750
1	0	1	1	71.10	80.37	65.30	62.77	77.76	**79.76**	79.22	76.34	-
0	1	1	1	72.99	**81.65**	66.03	63.00	80.32	79.72	79.22	**79.40**	3.250
1	1	1	1	72.99	**81.65**	66.03	62.99	80.32	79.72	79.22	**79.40**	-

nificant changes in classification quality. As described above, the distance weight was excluded from the test due to its minimal influence on the outcome, so the total number of compared approaches is 8. Therefore, F_F is distributed according to the F distribution with $8-1=7$ and $(8-1)(8-1)=49$ degrees of freedom, so for $\alpha = 0.05$, $F_F = 2.580 > F_{Critical} = 2.203$. However, the additional Nemenyi test did not reveal any significant differences for pairwise comparisons, neither for $\alpha = 0.05$ ($CD = 3.712$) nor for $\alpha = 0.10$ ($CD = 3.405$). The last column in Table 8.4 presents the average rank of each of the tested approaches (excluding distance).

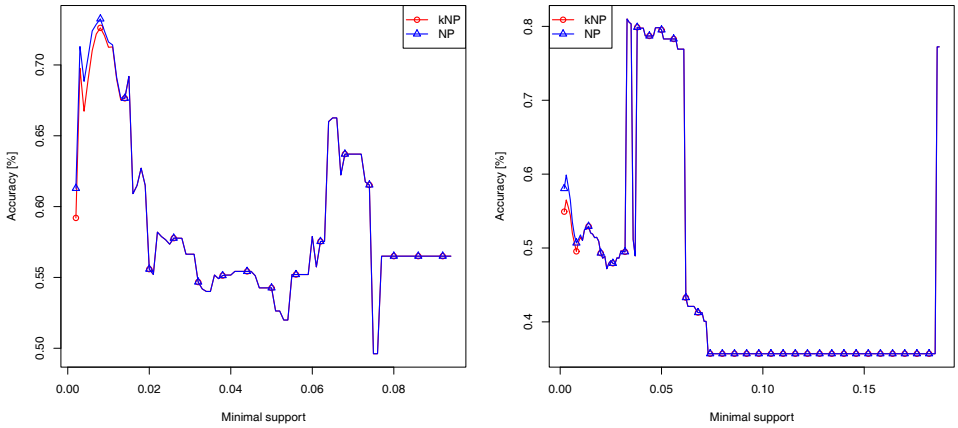
The results of this experiment show that weighting strategies significantly influence the quality of classification. However, the performed tests are insufficient to indicate which one works best. The lack of statistical significance in pairwise comparisons may be caused by insufficient number of datasets compared with the number of evaluated approaches, as we are testing 8 strategies on 8 datasets. Nevertheless, based on the ranking and the above discussion, we propose to weight votes using confidence and size, as these attributes performed best on 4/8 datasets and produced only slightly worse results on the remaining 4 datasets.

8.4.3 Parametrization

As stressed in the introduction of this thesis, proposing good parameter values plays an important role in performing any data mining activity. The following experiments aim at inspecting how the proposed approaches react to parameter changes. In these tests, both distance-based NP^r and instance-based k^+NP model were considered in order to evaluate all of the earlier discussed parameters: *minsup* and r for NP^r, and *minsup* and k for k^+NP. The first experiment illustrates how the minimal support parameter influences the classification quality. Since previous tests indicate a major difference in characteristics between ds and cs datasets, we performed this test using ds1 and cs1-2 datasets. The result is presented in Figures 8.4(a) and 8.4(b). Each plot illustrates the accuracy of both, NP^r and k^+NP approaches. The test was conducted with *minsup* changing from 0.001 with 0.001 step until pattern mining did not find any patterns for at least one of the classes.

The plots clearly indicate that minimal support can have a substantial influence on the classification quality. In case of both datasets, for lower *minsup* values there is a noticeable difference in quality between the instance- and distance-based approaches, the latter being always superior. However, after crossing a certain threshold both approaches start producing the same results. This is due to the fact that lower *minsup* produces more patterns and, thus, there are more patterns for the algorithms to chose from. As described earlier, these approaches were tested with $k = 1$ and $r = 0.2$, so this test in fact shows the *minsup* value after which either all patterns in the 0.2 range are in the same distance from the documents or no patterns fall into the 0.2 range.

This experiment reveals an issue common with most approaches based on frequent itemsets, i.e., sensitivity w.r.t. *minsup* parameter. However, it is important



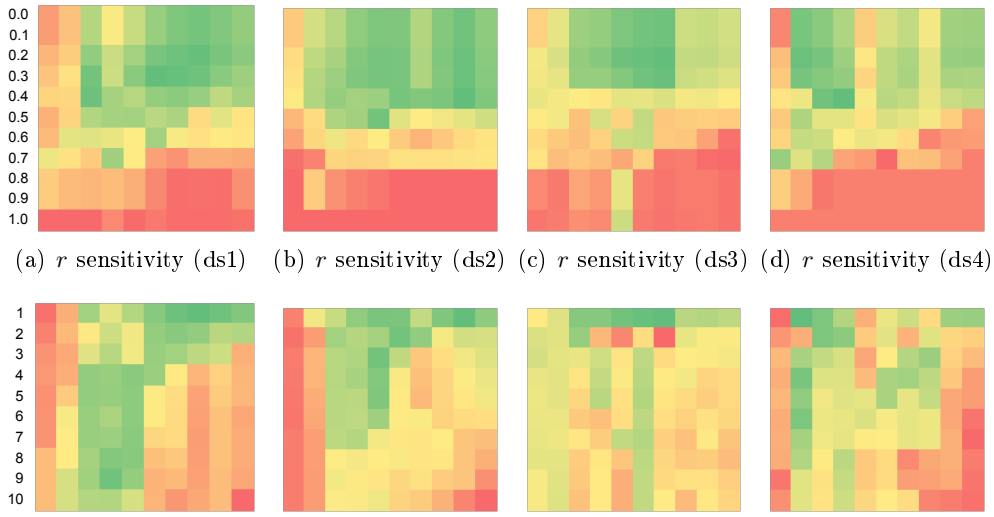
(a) *minsup* sensitivity (ds1) (b) *minsup* sensitivity (cs1-2)

Figure 8.4: Sensitivity w.r.t. *minsup*.

to note that in the case of classification this value is much easier to assess than in clustering described in the earlier chapters. Here for example, one can use the training dataset to tune the parameters with cross-validation [AC09].

Now, let us focus on the remaining parameters and examine how manipulating k and r influences the predictive capabilities of the discussed algorithms. Importantly, the effect these parameters have on classification outcome is dependent on minimal support, as different *minsup* values produce different pattern sets, not only in terms of tree structures but also in terms of quantity. That is why this test was performed with varying *minsup*, in order to include this factor in the analysis. The experiment was conducted on datasets ds1-4. Each dataset was tested for both model construction strategies (k^+NP and NP^r) with *minsup* changing from 0.001 to 0.010 with 0.001 step, r changing from 0 to 1 with 0.1 step, and k changing from 1 to 10 with 1 step. In order to facilitate the analysis of the results of this experiment, each test is illustrated with a heat map, with colors ranging from dark green — the highest quality, to dark red — the lowest quality. The results are presented in Figures 8.5(a)-8.5(h). In the diagrams, *minsup* increases horizontally while r and k change vertically.

The results of this experiment reveal noticeable differences in stability between the analyzed approaches. The heat maps for the distance-based approach (Figures 8.5(a)-8.5(d)) show that for r between 0 and 0.4, the algorithm produces stable, high quality results. After $r = 0.4$, the quality starts to degrade and drops significantly after the 0.8 threshold. More importantly, the plots show that *minsup* seems to have a relatively smaller effect on the quality of the distance-based model. In case of the instance-based approach (Figures 8.5(e)- 8.5(h)), the heat maps are fairly irregular and show that the approach performs reasonably stable w.r.t. *minsup* only for $k = 1$. There also seems to be a slight indication that the quality degrades diagonally towards the right bottom corner. This means that the higher the *minsup* the lower the k should be, and vice versa. This can be explained by the fact that higher minimal support values lead to fewer and more general patterns which, in turn, may overlap more. On the other hand, patterns



(a) r sensitivity (ds1) (b) r sensitivity (ds2) (c) r sensitivity (ds3) (d) r sensitivity (ds4)

(e) k sensitivity (ds1) (f) k sensitivity (ds2) (g) k sensitivity (ds3) (h) k sensitivity (ds4)

Figure 8.5: Sensitivity w.r.t. k and r.

with lower *minsup* are less common and more specific, so we need to consider more of them, in order to make a solid decision.

The tests reveal that changing the *minsup* value can have a high impact on the quality of both of the approaches. Furthermore, the experiments show that the distance-based model has a higher stability w.r.t. the r parameter than the instance-based model w.r.t. the k parameter with varying *minsup*. This, combined with the results from the previous section, suggests that the distance-based model is generally a favorable option, both in terms of ease of parametrization and classification quality.

8.4.4 Comparative analysis

The aim of the last experiment is to compare the proposed method with competitive solutions. kNP was compared with the classical rule-based classifier (Rule) and the state-of-the-art structural XML classification algorithm — XRules [ZA06]. The results for XRules were taken from [ZA06] while the results for the Rule classifier were obtained with our own implementation, described in the previous chapter (see Section 7.5). In the training phase, the training dataset is mined for maximal frequent subtrees separately for each class. Afterwards, the subtrees along with their corresponding classes form rules: $r = t_r \rightarrow c_r$, where t_r is a maximal frequent subtree and c_r is a class. Next, the rules are arranged into a ranking according to their confidence, support, and size. In the classification phase, each document is tested against each rule in the descending ranking order for subtree matching, and assigned to the class indicated by the first matching rule. If a document does not match any rule, it is assigned to a class based on a default rule (the majority class in the training dataset). The Accuracy of each algorithm is presented in Table 8.5. Additionally, next to Accuracy in column "Rule", there is a number illustrating the percentage of documents classified with a default rule.

Table 8.5: Comparison of kNP with other methods.

Algorithm	*Rule*	*XRules*	*kNP*
Dataset	Accuracy [%]		
ds1	64.54 (63.95%)	71.93	**73.25**
ds2	79.77 (58.06%)	79.77	**81.64**
ds3	56.77 (62.76%)	61.63	**66.35**
ds4	60.32 (65.81%)	**67.65**	64.04
cs1-2	80.37 (53.99%)	**83.63**	80.33
cs2-3	79.67 (58.77%)	**84.29**	79.72
cs3-1	79.16 (59.56%)	**84.39**	79.22
cs12-3	79.33 (57.75%)	**83.51**	79.40
Rank	2.813	**1.438**	1.750

The accuracies of the analyzed approaches show that both XRules and kNP are superior to the Rule algorithm. For $\alpha = 0.05$, with $3-1 = 2$ and $(3-1)(8-1) = 14$ degrees of freedom, the Friedman test indicates a statistically significant difference in quality between the analyzed algorithms: $F_F = 10.067 > F_{Critical} = 3.739$. With the ranks presented in the last row of Table 8.5, the additional Nemenyi test shows that, indeed, XRules and kNP perform significantly better than Rule: XRules for $\alpha = 0.05$ ($CD = 1.172$) and kNP for $\alpha = 0.10$ ($CD = 1.026$).

Comparing XRules with kNP we can see that XRules outperforms kNP in 5 out of 8 tests while kNP performs better in the other 3. Even though this result is inconclusive, we can observe that kNP performs better on balanced datasets ds$X$, while XRules excels on datasets with skewed class distributions. This observation indicates that enhancing the algorithms ability to handle unevenly distributed classes should be addressed in future research.

Looking at the results of the classical rule-based classifier (Rule), it is worth noticing that the default rule was used for over 50% of the documents. This illustrates the size of the default rule problem, addressed in this thesis, and is consistent with the observations from the previous chapter (see Section 7.5).

8.5 Conclusions

In this chapter, we have discussed a structural XML classification algorithm, called k-nearest patterns (kNP), inspired by the nearest neighbor and rule-based classifiers. The originality of the method stems from the combination of global information encapsulated in frequent patterns and local information available through pattern-document similarity measure — partial tree-edit distance. Several variations regarding the algorithm's components were discussed and experimentally evaluated. Finally, the proposed approach was tested for quality and parameter sensitivity, and compared with the classical rule-based classifier and the state-of-the-art XRules algorithm.

The experimental evaluation shows that the proposed approach produces results of competitive quality to XRules. Furthermore, kNP proved to be significantly better than the classical rule-based approach while additionally resolving the problem of excessive default rule usage.

As a future research direction, the proposed approach could be easily adapted to stream processing by incorporating an incremental frequent subtree mining algorithm, such as AdaTreeNat proposed by Bifet and Gavaldà [BG09]. To the best of our knowledge, currently the only XML stream classification algorithm is the one proposed by the aforementioned authors, thus, it would be very interesting to see how kNP performs in such a setting. So far, our preliminary experiments reveal that such an adoption produces very promising results [BP14]. Nevertheless, we will not discuss these results in detail as data streams constitute a separate research area and are out of the scope of this thesis.

kNP is the last approach discussed in this dissertation. In the next chapter, we will summarize the main findings of the thesis, confront them with the goals formed in the introduction and draw lines of future research.

9
Final conclusions and future work

In this thesis, we have discussed some of the main challenges of pattern-based clustering and classification of XML data and proposed new methods which successfully address these challenges. This chapter will summarize the main findings of this work and indicate possible lines of future research.

The analysis was divided into two core and one secondary domain, namely, clustering and classification, and approximate subtree matching. We described the main applications of these fields of study, outlined the main problems with the existing approaches, and explained why they are worth pursuing. Furthermore, we presented a comprehensive overview of the analyzed domains by discussing their components separately as well as examining the state-of-the-art approaches in each of the fields.

The first subject of inquiry was XML clustering. The domain overview revealed that most of the existing approaches rely on a local-information-oriented framework, what can lead to production of high quality, nevertheless poorly interpretable results. To address this issue, we proposed and formalized the XPattern framework — a generic methodology for clustering XML documents by patterns. The proposed solution allows to design algorithms based on global information and systematically ensures high interpretability of the achieved results thanks to the notion of profiles, summarizing the contents of clusters. This claim was validated with two working examples: a tree-based XCleaner2 algorithm and a path-based PathXP algorithm.

XCleaner2 represents documents as trees and defines patterns as maximal frequent subtrees. As trees encapsulate the whole information available in XML documents, the algorithm is capable of producing results of the highest quality even for more difficult datasets and was shown to match the state-of-the-art approach in this regard. The sensitivity tests showed that XCleaner2 is highly but predictably sensitive w.r.t. the minimum support (*minsup*) parameter — the lower the value the better the result, in general. However, as was shown in the experimental analysis, the costly processing of tree structures, especially for low *minsup* values, substantially lengthens the execution time. That is why,

XCleaner2 is recommended to be used with smaller documents or smaller datasets, when highly accurate measurements are required.

When bigger collections of large documents are being processed, PathXP — the second instance of XPattern — is recommended. PathXP decomposes the documents into paths and defines patterns as maximal frequent subpaths. Despite the information loss caused by this decomposition, the algorithm was able to match its main rivals in terms of quality.

On the example of PathXP, several additional experiments — concerning pattern-based clustering in general — were conducted. First, we compared alternative pattern definitions, namely: subtrees, paths of various lengths, tags, and metadata, and analyzed their applicability to different types of datasets. The experiment validated the previous observation that subtrees are only usable when narrow documents are being processed, as frequent subtree mining depends linearly on the height of trees but exponentially on their width. The comparison also confirms that frequent paths are able to produce results of high quality while maintaining reasonable processing time. However, further information reduction, i.e., limiting the length of frequent paths, negatively influences the clustering quality on average. In the extreme case, when path length was limited to 1 (tag-based approach), the results show that such patterns are only suitable for simple, highly heterogeneous datasets. When data homogeneity occurs, tags do not convey enough structural information to properly distinguish between different groups of documents. Finally, with patterns defined as simple metadata, e.g., number of distinct elements or height of a document tree, the results clearly indicate that such a representation is only capable of dealing with the simplest heterogeneous datasets and only compared to the tag-based approach. This fact, combined with a very high processing speed, makes this option usable only for preliminary analysis on massive datasets.

By analyzing various components of PathXP, we have shown that the often omitted information about the number of occurrences of a pattern in a single document can significantly improve the clustering quality ($\alpha = 0.05$). Moreover, limiting frequent paths to maximal only does not diminish the quality of the obtained results, yet, allows to create a more compact model. Somewhat surprising to find was that weighting patterns according to their uniqueness not only does not improve the quality, but actually significantly diminishes it ($\alpha = 0.05$).

The sensitivity tests show that PathXP, similarly as XCleaner2, is highly but predictably sensitive to the minimum support parameter and a simple heuristic based on the number of desired clusters was sufficient to accurately approximate this value. As a result, PathXP requires only a single parameter to be provided by user, i.e., number of clusters. To further adapt the proposed approach to real-world scenarios, where this value is often unknown, we proposed two heuristics which automatically detect the number of clusters: one for cases when some knowledge about the analyzed dataset is available and the other one when no preliminary assumptions are made. The experiments show that both methods provide a reasonable approximation of the number of clusters, however, intuitively, their predictive capabilities peak with easily separable groups of documents.

The second part of the work concentrated on the issue of XML classification. The literature overview shows that in addition to local-information-oriented solutions there are several approaches based on global information, most notably, the rule-based classifier, commonly used in this scenario. However, this approach suffers from the important default rule usage problem. The gravity of this issue was illustrated experimentally and the results revealed that in some cases it concerned up to 74% of documents. To resolve this issue, we introduced a pattern-document similarity measure, called partial tree-edit distance, which allows to blend some amount of local information into the otherwise globally-oriented scheme. The measure works as a combination of subtree matching and tree-edit distance and allows to measure the degree of containment of one tree in another. To be able to use partial tree-edit distance efficiently, we also put forward a dynamic programming algorithm, which calculates the proposed measure. The experiments show, that incorporating partial tree-edit distance into the basic rule-based classification scheme significantly improves the classification quality ($\alpha = 0.05$).

The developed measure served as a foundation for a new pattern-based XML classification algorithm, called k-nearest patterns. The algorithm was inspired by two classical approaches: rule-based and nearest neighbor classifier. This allows us to combine global information encapsulated in patterns with local information available through document-pattern similarity (partial tree-edit distance). The algorithm was shown to significantly outperform the classical rule-based approach ($\alpha = 0.10$) and produce results of quality competitive with the state-of-the-art method.

In addition to standard comparative analysis, we empirically analyzed several components of the proposed algorithm, including duplicate pattern treatment, alternative neighborhood definitions, and various vote weighting schemes. The analysis reveals that removing duplicate patterns from the model improves its predictive capabilities on average. Moreover, the distance-based neighborhood definition turned out to be equally good or better than any of the proposed instance-based models on all datasets. It was also shown to be more stable w.r.t. various parameter values than any of the alternatives. Finally, analyzing various vote weighting schemes revealed that the combination of patterns' size and their confidence performs best on average.

Based on the above, all research objectives stated in this thesis have been accomplished:

- We proposed and formalized a generic pattern-based framework for XML clustering, called XPattern.
- The framework was validated with two working algorithms: XCleaner2 and PathXP.
- We analyzed alternative pattern definitions, namely: subtrees, paths of various lengths, tags, and metadata, in terms of their applicability to different types of datasets.
- We proposed and validated a pattern-based algorithm for XML classification, called k-nearest patterns.

- We proposed a measure for evaluating similarity between pattern trees and document trees, called partial tree-edit distance, along with an efficient algorithm for calculating this measure.
- All proposed algorithms were experimentally evaluated with the emphasis on sensitivity w.r.t. their parameters.

Let us now discuss some of the possible lines of future work emerging from the research conducted in this thesis. The literature overview already highlights several open issues in this field. Firstly, a strong dependency between the used similarity measures and document representations can be noticed. Similarity between documents represented with paths is usually evaluated with simple occurrence counting, while tree structures are compared using tree-edit distance measures. Consequently, there is a room for developing more complex path measures which would take into account not only their presence but also their structural relationships. Furthermore, when analyzing the clustering approaches based on the traditional, three-step framework (document transformation, similarity evaluation, document clustering), one can observe that the last step of the process leaves room for further developments, as currently only general, non-XML-specific clustering algorithms are in use.

A different take on future work can be explored in the context of classification. The research conducted in this thesis focused on processing of large, static collections of XML data. Nowadays however, the data processing paradigm is shifting from static to streaming data, where documents have to be processed online using limited memory. As most existing XML classifiers are capable of processing only static data, there is a need to develop new approaches dedicated for streaming environments. Example applications involving processing of XML data generated at high rates include monitoring messages exchanged between web-services, event stream management, distributed ETL processes, and services for RSS feeds [MP09]. The k-nearest patterns algorithm, proposed in this thesis, could be easily adapted to stream processing by incorporating an incremental frequent subtree mining algorithm, such as AdaTreeNat proposed by Bifet and Gavaldà [BG09]. Our preliminary experiments with this concept show very promising results [BP14]. Currently, the only XML classifier designed for stream processing is the one proposed by the aforementioned authors. Therefore, this issue opens many possibilities for novel research.

In the introduction, we have also touched on the matter of data distribution and its influence on the data mining process. In our experiments, we used datasets of various distributions. However, we did not discuss datasets with highly skewed distributions, e.g., one class containing 90% of documents. Such datasets are called imbalanced and the issue of processing such data has grown to constitute a separate research area. As the approaches proposed in this thesis use patterns defined in terms of frequency across the dataset, imbalanced data can have a significant impact on the quality of our methods. Inasmuch as this issue is to some extent addressed in classification, as patterns are mined separately for each class, it is potentially problematic in the clustering task.

When developing a solution in the XML mining domain, researchers face another problem, completely unrelated with the scientific task — shortage of publicly available, real-world datasets. Such a situation makes it very difficult to analyze newly proposed methods, not to mention compare them with existing approaches. The comparison is even more difficult to perform since usually not only the datasets are unavailable but also the implementations of the competitive algorithms. This situation partially stems from the fact that, even though XML clustering has many applications, the actual application rate of the proposed methods is very low. Clearly, there is a strong need for some kind of XML clustering platform or a repository where researchers could store and test their algorithms and analysts could post real datasets and process them with different methods. This would greatly facilitate the comparison of existing methods and allow easy access to the newest solutions to real XML-related problems. This could be achieved either by a separate tool/platform or a set of extensions to existing data mining tools, e.g., R [R C13], Weka [HFH+09], or RapidMiner [KMF06]. Currently, neither publicly available research tools nor commercial systems dedicated for XML clustering by structure exist. We believe that addressing this problem would be of high practical value to the research community and could improve the dissemination of newly developed approaches.

A
Subtree Matching Algorithm

In order to check if a document contains a pattern, a subtree matching algorithm was implemented. The algorithm iterates through the elements of a string encoded document and tries to match pattern elements to the document's tree nodes. For each pair of compared elements we check if the current document and pattern labels are identical and if the relative document and pattern depth levels are equal. Additionally, in each step the algorithm checks if it has traversed a level higher than the searched element's parent. If so, it is forced to search once again for the last matched element. The exact steps of the procedure are listed in Algorithm 7.

In the worst case scenario, the algorithm attempts to match a subtree beginning at every node in all of the documents in the dataset. Since the algorithm restarts from every possible pattern root found in a document, for a single document and pattern, it requires $O(|d|^2)$ time. For a dataset of n documents, a set of p patterns, and with l being the number of nodes in the longest document in the dataset, the pessimistic complexity of finding all pattern-document connections in is $O(npl^2)$.

Let us analyze how the algorithm operates with a simple example depicted in Figure A.1. Let d be an XML document and p be a pattern. We begin with transforming p from its string format (2 3 4 -1 -1) to a $label[level]$ format (2[1] 3[2] 4[3]). Next, we find the first occurrence of the pattern's root and search for the next pattern element (Figure A.1(a)). While searching for the second pattern element we find an element with a label identical to that of the pattern's root, so we set it as *startPos* (Figure A.1(b)). The *startPos* variable allows us to remember the next root occurrence if we do not manage to find the pattern in a single pass. Continuing with the depth-first traversal of the document we go a level higher than the searched element's parent (Figure A.1(c)). This forces us to search once again for the root of the pattern. In Figures A.1(d-f) we see another attempt at matching p to d, this time from $docPos = 11$. While searching for the third pattern element we reach the level of its parent and have to search for it again. Figures A.1(g-i) show the last three steps in matching pattern p to document d.

Algorithm 7 Subtree Matching Algorithm

Require: an XML document d and a pattern p in a string format
Ensure: *true* if p is a subtree of d, *false* otherwise

1: $docPos \leftarrow 0$;
2: $startPos \leftarrow null$
3: calculate depth level for each label $p_i \in p$ and remove all return marks -1 from p;
4: $docLevel \leftarrow 0$;
5: **if** $startPos \neq null$ **then**
6: $docPos \leftarrow startPos$;
7: **end if**
8: **if** $startPos >$ last occurrence of $p[0]$ in d **then**
9: **return** *false*;
10: **end if**
11: $startPos \leftarrow null$;
12: **for** $i = \{0, ..., legth(p) - 1\}$ **do**
13: **while** $docPos < docLength$ and ($p[i] \neq d[docPos]$ or $docLevel \neq depth(p[i])$) **do**
14: **if** $i > 0$ **then**
15: **if** $d[docPos] \neq -1$ **then**
16: $docLevel \leftarrow docLevel + 1.$;
17: **else**
18: $docLevel \leftarrow docLevel - 1$;
19: **if** $IsLevelCrossed(p, docLevel, i)$ **then**
20: $i \leftarrow$ index of the parent of $p[i]$;
21: **end if**
22: **end if**
23: **if** $d[docPos] = p[0]$ and $startPos = null$ **then**
24: $startPos \leftarrow docPos$;
25: **end if**
26: **end if**
27: $docPos \leftarrow docPos + 1$;
28: **end while**
29: **if** $d[docPos] = p[0]$ and $startPos = null$ **then**
30: $startPos \leftarrow docPos$;
31: **end if**
32: **if** $docPos > docLength$ **then**
33: **goto** 5;
34: **else if** $i = legth(p) - 1$ **then**
35: **return** *true*;
36: **else**
37: $docLevel \leftarrow docLevel + 1$;
38: $docPos \leftarrow docPos + 1$;
39: **end if**
40: **end for**

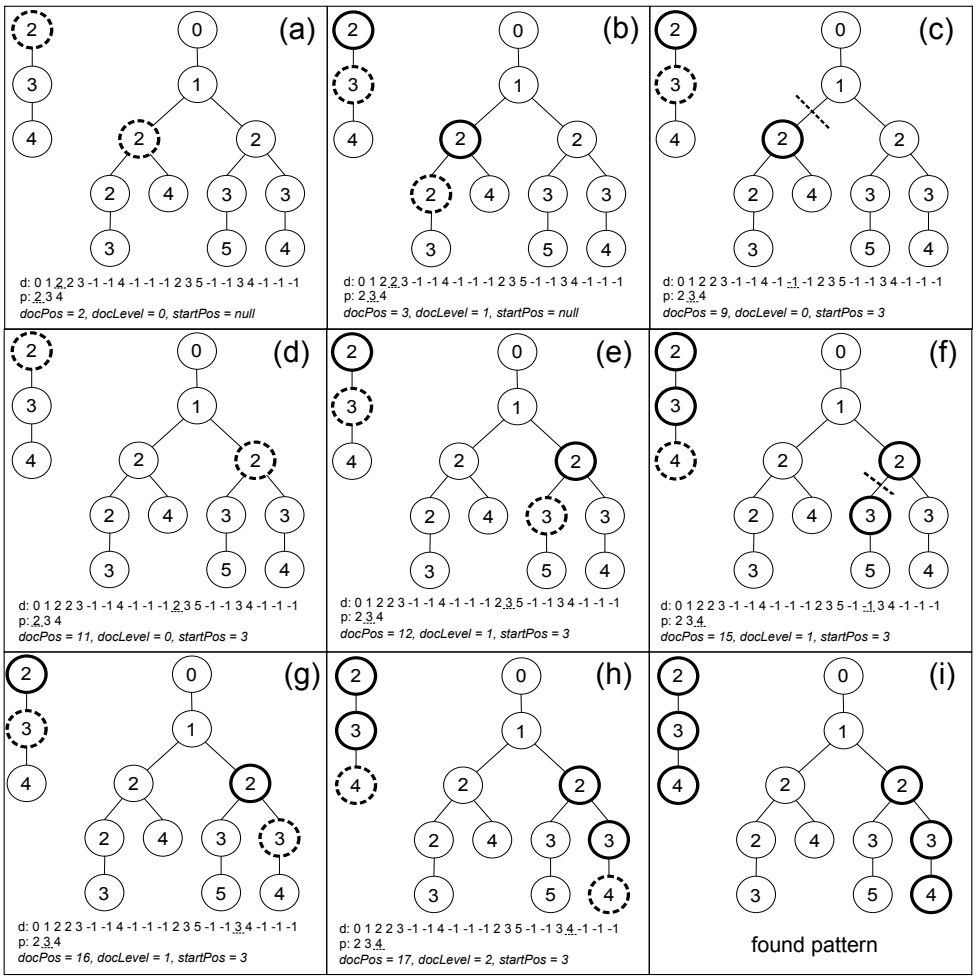


Figure A.1: Subtree matching example.

Bibliography

[AAWS09] Bill Andreopoulos, Aijun An, Xiaogang Wang, and Michael Schroeder. A roadmap of clustering algorithms: finding a match for a biomedical application. *Briefings in Bioinformatics*, 10(3):297–314, 2009.

[ABBP10] Nikolaus Augsten, Denilson Barbosa, Michael H. Böhlen, and Themis Palpanas. TASM: top-k approximate subtree matching. In *Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*, pages 353–364. IEEE, 2010.

[ABG05] Nikolaus Augsten, Michael H. Böhlen, and Johann Gamper. Approximate matching of hierarchical data using pq-grams. In *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, pages 301–312. ACM, 2005.

[ABMS12] Ali Aïtelhadj, Mohand Boughanem, Mohamed Mezghiche, and Fatiha Souam. Using structural similarity for clustering XML documents. *Knowledge and Information Systems*, 32(1):109–139, 2012.

[AC09] Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection. *ArXiv e-prints*, July 2009.

[ACS02] Sihem Amer-Yahia, SungRan Cho, and Divesh Srivastava. Tree pattern relaxation. In *Advances in Database Technology - EDBT 2002, 8th International Conference on Extending Database Technology, Prague, Czech Republic, March 25-27, Proceedings*, volume 2287 of *Lecture Notes in Computer Science*, pages 496–513. Springer, 2002.

[AMNS11] Alsayed Algergawy, Marco Mesiti, Richi Nayak, and Gunter Saake. XML data clustering: An overview. *ACM Computing Surveys*, 43(4):25, 2011.

[AMT08] Panagiotis Antonellis, Christos Makris, and Nikos Tsirakis. Xedge: clustering homogeneous and heterogeneous XML documents using edge summaries. In *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC), Fortaleza, Ceara, Brazil, March 16-20, 2008*, pages 1081–1088. ACM, 2008.

[ANA10] Mohamad Alishahi, Mahmoud Naghibzadeh, and Baharak Shakeri Aski. Tag name structure-based clustering of XML documents. *International Journal of Electrical and Computer Engineering*, 2(1):119–126, February 2010.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 487–499. Morgan Kaufmann, 1994.

[ATW+07] Charu C. Aggarwal, Na Ta, Jianyong Wang, Jianhua Feng, and Mohammed Javeed Zaki. Xproj: a framework for projected structural clustering of xml documents. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, California, USA, August 12-15, 2007*, pages 46–55. ACM, 2007.

[BG09] Albert Bifet and Ricard Gavaldà. Adaptive XML tree classification on evolving data streams. In *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2009, Bled, Slovenia, September 7-11, 2009, Proceedings, Part I*, volume 5781 of *Lecture Notes in Computer Science*, pages 147–162. Springer, 2009.

[BH07] Abdelhamid Bouchachia and Marcus Hassler. Classification of XML documents. In *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2007, part of the IEEE Symposium Series on Computational Intelligence 2007, Honolulu, Hawaii, USA, 1-5 April 2007*, pages 390–396. IEEE, 2007.

[Bil05] Philip Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337(1-3):217–239, 2005.

[BLMP11] Dariusz Brzezinski, Anna Lesniewska, Tadeusz Morzy, and Maciej Piernik. XCleaner: A new method for clustering XML documents by structure. *Control and Cybernetics*, 40(3):877–891, 2011.

[BMAD06] Z. Bakar, R. Mohemad, A. Ahmad, and M. Deris. A Comparative Study for Outlier Detection Techniques in Data Mining. In *2006 IEEE Conference on Cybernetics and Intelligent Systems*, pages 1–6. IEEE, November 2006.

[BMKL02] Denilson Barbosa, Alberto O. Mendelzon, John Keenleyside, and Kelly A. Lyons. Toxgene: a template-based data generator for XML.

In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, June 3-6, 2002*, page 616. ACM, 2002.

[BP14] Dariusz Brzezinski and Maciej Piernik. Adaptive XML stream classification using partial tree-edit distance. In *Foundations of Intelligent Systems - 21st International Symposium, ISMIS 2014, Roskilde, Denmark, June 25-27, 2014. Proceedings*, volume 8502 of *Lecture Notes in Computer Science*, pages 10–19. Springer, 2014.

[BPSM98] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. Recommendation, World Wide Web Consortium (W3C), 1998.

[BR99] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[BSM97] Bioinformatic Sequence Markup Language - BSML. `http://xml.coverpages.org/bsml.html`, 1997.

[But04] David Buttler. A short survey of document structure similarity algorithms. In *Proceedings of the International Conference on Internet Computing, IC '04, Las Vegas, Nevada, USA, June 21-24, 2004, Volume 1*, pages 3–9. CSREA Press, 2004.

[Cha99] Sudarshan S. Chawathe. Comparing hierarchical data in external memory. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 90–101. Morgan Kaufmann, 1999.

[CMK07] Il-Hwan Choi, Bongki Moon, and Hyoung-Joo Kim. A clustering method based on path similarities of XML data. *Data & Knowledge Engineering*, 60(2):361–376, 2007.

[CML95] Chemical Markup Language - CML. `http://www.xml-cml.org/`, 1995.

[CMOT04] Gianni Costa, Giuseppe Manco, Riccardo Ortale, and Andrea Tagarelli. A tree-based approach to clustering XML documents by structure. In *Knowledge Discovery in Databases: PKDD 2004, 8th European Conference on Principles and Practice of Knowledge Discovery in Databases, Pisa, Italy, September 20-24, 2004, Proceedings*, volume 3202 of *Lecture Notes in Computer Science*, pages 137–148. Springer, 2004.

[CO14] Sara Cohen and Nerya Or. A general algorithm for subtree similarity-search. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pages 928–939. IEEE, 2014.

[Coh13] Sara Cohen. Indexing for subtree similarity-search using edit distance. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 49–60. ACM, 2013.

[COR13] Gianni Costa, Riccardo Ortale, and Ettore Ritacco. X-class: Associative classification of XML documents by structure. *ACM Transactions on Information Systems*, 31(1):3:1–3:40, 2013.

[CRGW96] Sudarshan S. Chawathe, Anand Rajaraman, Hector Garcia-Molina, and Jennifer Widom. Change detection in hierarchically structured information. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996.*, pages 493–504. ACM Press, 1996.

[CTT05] Laurent Candillier, Isabelle Tellier, and Fabien Torre. Transforming XML trees for efficient classification and clustering. In *Advances in XML Information Retrieval and Evaluation, 4th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2005, Dagstuhl Castle, Germany, November 28-30, 2005, Revised Selected Papers*, volume 3977 of *Lecture Notes in Computer Science*, pages 469–480. Springer, 2005.

[CXYM05] Yun Chi, Yi Xia, Yirong Yang, and Richard R. Muntz. Mining closed and maximal frequent subtrees from databases of labeled rooted trees. *IEEE Trans. Knowl. Data Eng.*, 17(2):190–202, 2005.

[DA02] Antoine Doucet and Helena Ahonen-Myka. Naïve clustering of a large XML document collection. In *Proceedings of the First Workshop of the INitiative for the Evaluation of XML Retrieval (INEX), Schloss Dagstuhl, Germany, December 9-11, 2002*, pages 81–87, 2002.

[DCWS06] Theodore Dalamagas, Tao Cheng, Klaas-Jan Winkel, and Timos K. Sellis. A methodology for clustering XML documents by structure. *Information Systems*, 31(3):187–228, 2006.

[Dem06] Janez Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[DG04] Ludovic Denoyer and Patrick Gallinari. Bayesian network model for semi-structured document classification. *Information Processing & Management*, 40(5):807–827, 2004.

[DG07] Ludovic Denoyer and Patrick Gallinari. Report on the XML mining track at INEX 2005 and INEX 2006: categorization and clustering of XML documents. *SIGIR Forum*, 41(1):79–90, 2007.

[DMRW09] Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An optimal decomposition algorithm for tree edit distance. *ACM Transactions on Algorithms*, 6(1):2:1–2:19, 2009.

[FMM+05] Sergio Flesca, Giuseppe Manco, Elio Masciari, Luigi Pontieri, and Andrea Pugliese. Fast detection of XML structural similarity. *IEEE Transactions on Knowledge and Data Engineering*, 17(2):160–175, 2005.

[GMS07] Giovanna Guerrini, Marco Mesiti, and Ismael Sanz. *Web Data Management Practices: Emerging Techniques and Technologies*, chapter 3. An Overview of Similarity Measures for Clustering XML Documents. Idea Group Inc (IGI), 2007.

[GMT05] Calin Garboni, Florent Masseglia, and Brigitte Trousse. Sequential pattern mining for structure-based XML document classification. In *Advances in XML Information Retrieval and Evaluation, 4th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2005, Dagstuhl Castle, Germany, November 28-30, 2005, Revised Selected Papers*, volume 3977 of *Lecture Notes in Computer Science*, pages 458–468. Springer, 2005.

[GRS00] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. ROCK: A robust clustering algorithm for categorical attributes. *Information Systems*, 25(5):345–366, 2000.

[GW97] Roy Goldman and Jennifer Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 436–445. Morgan Kaufmann, 1997.

[HAB12] Sven Helmer, Nikolaus Augsten, and Michael H. Böhlen. Measuring structural similarity of semistructured data based on information-theoretic approaches. *The VLDB Journal*, 21(5):677–702, 2012.

[HD13] Marouane Hachicha and Jérôme Darmont. A survey of XML tree patterns. *IEEE Transactions on Knowledge and Data Engineering*, 25(1):29–46, 2013.

[Hel07] Sven Helmer. Measuring the structural similarity of semistructured documents using entropy. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pages 1022–1032. ACM, 2007.

[HFH+09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explorations*, 11(1), 2009.

[HFS+03] Michael Hucka, Andrew Finney, Herbert M. Sauro, H. Bolouri, John C. Doyle, Hiroaki Kitano, Adam P. Arkin, Benjamin J. Bornstein, D. Bray, A. Cornish-Bowden, Autumn A. Cuellar, Serge Dronov, Ernst Dieter Gilles, Martin Ginkel, Victoria Gor, Igor

Goryanin, W. J. Hedley, T. Charles Hodgman, J. H. Hofmeyr, Peter J. Hunter, Nick S. Juty, J. L. Kasberger, Andreas Kremling, Ursula Kummer, Nicolas Le Novère, Leslie M. Loew, D. Lucio, Pedro Mendes, E. Minch, Eric Mjolsness, Yoichi Nakayama, M. R. Nelson, Poul M. F. Nielsen, T. Sakurada, James C. Schaff, Bruce E. Shapiro, Thomas Simon Shimizu, Hugh D. Spence, Jörg Stelling, Koichi Takahashi, Masaru Tomita, J. Wagner, and J. Wang. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, 2003.

[HNP09] Alon Y. Halevy, Peter Norvig, and Fernando Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12, 2009.

[HPRS07] Dušan Husek, Jaroslav Pokorny, Hana Rezankova, and Václav Snasel. *Web Data Management Practices: Emerging Techniques and Technologies*, chapter 1. Data Clustering: From Documents to the Web. Idea Group Inc (IGI), 2007.

[HST+05] Markus Hagenbuchner, Alessandro Sperduti, Ah Chung Tsoi, Francesca Trentini, Franco Scarselli, and Marco Gori. Clustering XML documents using self-organizing maps for structures. In *Advances in XML Information Retrieval and Evaluation, 4th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2005, Dagstuhl Castle, Germany, November 28-30, 2005, Revised Selected Papers*, volume 3977 of *Lecture Notes in Computer Science*, pages 481–496. Springer, 2005.

[HZL02] Daniel Hanisch, Ralf Zimmer, and Thomas Lengauer. ProML - the protein markup language for specification of protein sequences, structures and families. *In Silico Biol*, 2(3):313–24, 2002.

[Kle98] Philip N. Klein. Computing the edit-distance between unrooted ordered trees. In *Algorithms - ESA '98, 6th Annual European Symposium, Venice, Italy, August 24-26, 1998, Proceedings*, volume 1461 of *Lecture Notes in Computer Science*, pages 91–102. Springer, 1998.

[KMF06] Ralf Klinkenberg, Ingo Mierswa, and Simon Fischer. Rapid miner. `https://rapidminer.com/`, 2006.

[Koh89] T. Kohonen. *Self-organization and associative memory: 3rd edition*. Springer-Verlag New York, Inc., 1989.

[KSC02] Lukasz A. Kurgan, Waldemar Swiercz, and Krzysztof J. Cios. Semantic mapping of XML tags using inductive machine learning. In *Proceedings of the 2002 International Conference on Machine Learning and Applications - ICMLA 2002, June 24-27, 2002, Las Vegas, Nevada, USA.*, pages 99–109. CSREA Press, 2002.

[KTNL07] Sangeetha Kutty, Tien Tran, Richi Nayak, and Yuefeng Li. Clustering XML documents using closed frequent subtrees: A structural similarity approach. In *Focused Access to XML Documents, 6th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2007, Dagstuhl Castle, Germany, December 17-19, 2007. Selected Papers*, volume 4862 of *Lecture Notes in Computer Science*, pages 183–194. Springer, 2007.

[LCCL05] Ho-pong Leung, Korris Fu-Lai Chung, Stephen Chi-fai Chan, and Robert Wing Pong Luk. XML document clustering using common xpath. In *2005 International Workshop on Challenges in Web Information Retrieval and Integration (WIRI 2005), 8-9 April 2005, Tokyo, Japan*, pages 91–96. IEEE Computer Society, 2005.

[LCMY04] Wang Lian, David Wai-Lok Cheung, Nikos Mamoulis, and Siu-Ming Yiu. An efficient and scalable algorithm for clustering XML documents by structure. *IEEE Transactions on Knowledge and Data Engineering*, 16(1):82–96, 2004.

[LSG14] Issam H. Laradji, Mohammed Salahadin, and Lahouari Ghouti. XML classification using ensemble learning on extracted features. In *Proceedings of the 2014 ACM Southeast Regional Conference, Kennesaw, GA, USA, March 28 - 29, 2014*, page 1. ACM, 2014.

[LXJZ14] Jie Li, Zheng Xu, Yayun Jiang, and Rui Zhang. The overview of big data storage and management. In *IEEE 13th International Conference on Cognitive Informatics and Cognitive Computing, ICCI*CC 2014, London, UK, August 18-20, 2014*, pages 510–513. IEEE, 2014.

[Moo96] Todd K. Moon. The expectation-maximization algorithm. *IEEE Signal Processing Magazine*, 13(6):47–60, November 1996.

[MP09] Veronica Mayorga and Neoklis Polyzotis. Sketch-based summarization of ordered XML streams. In *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, pages 541–552. IEEE, 2009.

[Nay08] Richi Nayak. Fast and effective clustering of XML data using structural information. *Knowledge and Information Systems*, 14(2):197–215, 2008.

[NI06] Richi Nayak and Wina Iryadi. Xmine: A methodology for mining XML structure. In *Frontiers of WWW Research and Development - APWeb 2006, 8th Asia-Pacific Web Conference, Harbin, China, January 16-18, 2006, Proceedings*, volume 3841 of *Lecture Notes in Computer Science*, pages 786–792. Springer, 2006.

[NJ02] Andrew Nierman and H. V. Jagadish. Evaluating structural similarity in XML documents. In *Proceedings of the Fifth International*

Workshop on the Web and Databases, WebDB 2002, pages 61–66. University of California, 2002.

[NVK+09] Richi Nayak, Christopher M. De Vries, Sangeetha Kutty, Shlomo Geva, Ludovic Denoyer, and Patrick Gallinari. Overview of the INEX 2009 XML mining track: Clustering and classification of XML documents. In *Focused Retrieval and Evaluation, 8th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2009, Brisbane, Australia, December 7-9, 2009, Revised and Selected Papers*, volume 6203 of *Lecture Notes in Computer Science*, pages 366–378. Springer, 2009.

[PA11] Mateusz Pawlik and Nikolaus Augsten. RTED: A robust algorithm for the tree edit distance. *PVLDB*, 5(4):334–345, 2011.

[PBM15] Maciej Piernik, Dariusz Brzezinski, and Tadeusz Morzy. Clustering xml documents by patterns. *Knowledge and Information Systems*, 2015.

[PBML14] Maciej Piernik, Dariusz Brzezinski, Tadeusz Morzy, and Anna Lesniewska. XML clustering: A review of structural approaches. *The Knowledge Engineering Review*, 2014.

[Pos09] Thomas Pospech. *GML - Geography Markup Language*. GRIN Verlag, May 2009.

[R C13] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. ISBN 3-900051-07-0.

[Ran71] William M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971.

[RMS06] Davood Rafiei, Daniel L. Moise, and Dabo Sun. Finding syntactic similarities between XML documents. In *17th International Workshop on Database and Expert Systems Applications (DEXA 2006), 4-8 September 2006, Krakow, Poland*, pages 512–516. IEEE Computer Society, 2006.

[Rou87] Peter Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20(1):53–65, November 1987.

[SB88] Gerard Salton and Chris Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988.

[Sel77] Stanley M. Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6(6):184–186, 1977.

[SIG11] SIGMOD. http://www.sigmod.org/publications/sigmod-record/xml-edition, August 2011.

[SM02] Torsten Schlieder and Holger Meuss. Querying and ranking XML documents. *Journal of the Association for Information Science and Technology*, 53(6):489–503, 2002.

[SW03] Yun Shen and Bing Wang. Clustering schemaless XML documents. In *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE - OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003*, volume 2888 of *Lecture Notes in Computer Science*, pages 767–784. Springer, 2003.

[Tai79] Kuo-Chung Tai. The tree-to-tree correction problem. *Journal of the ACM*, 26(3):422–433, 1979.

[TC12] Joe Tekli and Richard Chbeir. A novel XML document structure comparison framework based-on sub-tree commonalities and label semantics. *Journal of Web Semantics*, 11:14–40, 2012.

[TCY07] Joe Tekli, Richard Chbeir, and Kokou Yétongnon. Efficient XML structural similarity detection using sub-tree commonalities. In *XXII Simpósio Brasileiro de Banco de Dados, 15-19 de Outubro, João Pessoa, Paraíba, Brasil, Anais*, pages 116–130. SBC, 2007.

[TCY09] Joe Tekli, Richard Chbeir, and Kokou Yetongnon. Survey: An overview on XML similarity: Background, current trends and future directions. *Computer Science Review*, 3(3):151–173, August 2009.

[TNB07] Tien Tran, Richi Nayak, and Peter Bruza. Document clustering using incremental and pairwise approaches. In *Focused Access to XML Documents, 6th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2007, Dagstuhl Castle, Germany, December 17-19, 2007. Selected Papers*, volume 4862 of *Lecture Notes in Computer Science*, pages 222–233. Springer, 2007.

[TSK05] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.

[TSW03] Martin Theobald, Ralf Schenkel, and Gerhard Weikum. Exploiting structure, annotation, and ontological knowledge for automatic classification of XML data. In *International Workshop on Web and Databases, San Diego, California, June 12-13, 2003*, pages 1–6, 2003.

[VFGL05] Anne-Marie Vercoustre, Mounir Fegas, Saba Gul, and Yves Lechevallier. A flexible structured-based representation for XML document mining. In *Advances in XML Information Retrieval and Evaluation, 4th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2005, Dagstuhl Castle, Germany, November*

28-30, 2005, Revised Selected Papers, volume 3977 of *Lecture Notes in Computer Science*, pages 443–457. Springer, 2005.

[VMB08] Waraporn Viyanon, Sanjay Kumar Madria, and Sourav S. Bhowmick. XML data integration based on content and structure similarity using keys. In *On the Move to Meaningful Internet Systems: OTM 2008, OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008, Monterrey, Mexico, November 9-14, 2008, Proceedings, Part I*, volume 5331 of *Lecture Notes in Computer Science*, pages 484–493. Springer, 2008.

[VNK+10] Christopher M. De Vries, Richi Nayak, Sangeetha Kutty, Shlomo Geva, and Andrea Tagarelli. Overview of the INEX 2010 XML mining track: Clustering and classification of XML documents. In *Comparative Evaluation of Focused Retrieval - 9th International Workshop of the Inititative for the Evaluation of XML Retrieval, INEX 2010, Vugh, The Netherlands, December 13-15, 2010, Revised Selected Papers*, volume 6932 of *Lecture Notes in Computer Science*, pages 363–376. Springer, 2010.

[VPA07] Athena Vakali, George Pallis, and Lefteris Angelis. *Web Data Management Practices: Emerging Techniques and Technologies*, chapter 2. Clustering Web Information Services. Idea Group Inc (IGI), 2007.

[VPD04] Athena Vakali, Jaroslav Pokorný, and Theodore Dalamagas. An overview of web data clustering practices. In *Current Trends in Database Technology - EDBT 2004 Workshops, EDBT 2004 Workshops PhD, DataX, PIM, P2P&DB, and ClustWeb, Heraklion, Crete, Greece, March 14-18, 2004, Revised Selected Papers*, volume 3268 of *Lecture Notes in Computer Science*, pages 597–606. Springer, 2004.

[W3C95] The World Wide Web Consortium — W3C. `http://www.w3.org/`, 1995.

[Wil45] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[WIN+05] John D. Westbrook, Nobutoshi Ito, Haruki Nakamura, Kim Henrick, and Helen M. Berman. Pdbml: the representation of archival macromolecular structure data in xml. *Bioinformatics*, 21(7):988–992, 2005.

[WWZ+15] Yue Wang, Hongzhi Wang, Liyan Zhang, Yang Wang, Jianzhong Li, and Hong Gao. Extend tree edit distance for effective object identification. *Knowledge and Information Systems*, pages 1–28, 2015.

[Yan04] Guizhen Yang. The complexity of mining maximal frequent itemsets and maximal frequent patterns. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*, pages 344–353. ACM, 2004.

[YKT05] Rui Yang, Panos Kalnis, and Anthony K. H. Tung. Similarity evaluation on tree-structured data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, pages 754–765. ACM, 2005.

[YW09] Jianwu Yang and Songlin Wang. Extended VSM for XML document classification using frequent subtrees. In *Focused Retrieval and Evaluation, 8th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2009, Brisbane, Australia, December 7-9, 2009, Revised and Selected Papers*, volume 6203 of *Lecture Notes in Computer Science*, pages 441–448. Springer, 2009.

[ZA06] Mohammed Javeed Zaki and Charu C. Aggarwal. Xrules: An effective algorithm for structural classification of XML data. *Machine Learning*, 62(1-2):137–170, 2006.

[Zak02] Mohammed Javeed Zaki. Efficiently mining frequent trees in a forest. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada*, pages 71–80. ACM, 2002.

[ZJS10] Ying-wen Zhu, Gen-Lin Ji, and Qin-hong Sun. Clustering GML documents using maximal frequent induced subtrees. In *Seventh International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2010, 10-12 August 2010, Yantai, Shandong, China*, pages 2265–2269. IEEE, 2010.

[ZK02] Ying Zhao and George Karypis. Evaluation of hierarchical clustering algorithms for document datasets. In *Proceedings of the 2002 ACM CIKM International Conference on Information and Knowledge Management, McLean, VA, USA, November 4-9, 2002*, pages 515–524. ACM, 2002.

[ZLZC11] Yan-Tao Zheng, Yiqun Li, Zheng-Jun Zha, and Tat-Seng Chua. Mining travel patterns from gps-tagged photos. In *Advances in Multimedia Modeling - 17th International Multimedia Modeling Conference, MMM 2011, Taipei, Taiwan, January 5-7, 2011, Proceedings, Part I*, volume 6523 of *Lecture Notes in Computer Science*, pages 262–272. Springer, 2011.

[ZS89] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262, 1989.

[ZSW94] Kaizhong Zhang, Dennis Shasha, and Jason Tsong-Li Wang. Approximate tree matching in the presence of variable length don't cares. *Journal of Algorithms*, 16(1):33–66, 1994.

[ZWB+11] Xiangguo Zhao, Guoren Wang, Xin Bi, Peizhen Gong, and Yuhai Zhao. XML document classification based on ELM. *Neurocomputing*, 74(16):2444–2451, 2011.

Index

A

Accuracy, 28
- equal, 29
- proportional, 29

B

Between group sum of squares, SSB, 29

C

centroid, 24
classification, 25
- k-nearest neighbors, kNN, 27
- rule-based, 25

classification model, 25
classifier, 25
- lazy classifier, 27

cluster, 21
clustering, 21
- hierarchical, 22
  - agglomerative, AHC, 22
  - average-link, 23
  - complete-link, 23
  - divisive, 22
  - single-link, 23
- partition-based, 23
  - k-means, 24
  - k-medoids, 24

connection, 44
- connection strength, 44

containment, 44

D

data
- labeled, 25
- training, 25
- unlabeled, 25

default rule, 26
dendrogram, 22

E

external measures, 28

F

F-score, 28
false negative, FN, 28
false positive, FP, 28
feature, 44
- frequent, 48
- maximal, 48

forest, 18, 78
full path, 14

I

information
- global, 5
- local, 5

internal measures, 29
- cohesion, 29
- isolation, 29

K

k-nearest patterns, kNP, 89

keyroots, 19

L
label, 12

M
model construction strategies
  Leave all, 90
  Remove duplicates, 90
  Remove embedded, 91

N
neighborhood definition
  distance-based, 93
  instance-based, 93

P
partial mapping, 80
partial tree-edit distance, PTED, 77, 80
partial tree-edit sequence, 80
PathXP, 57
pattern, 44
  confidence, 91
  size, 91
  support, 91
Precision, 28
profile
  (classification), 90
  (clustering), 44
Purity, 29

R
Rand Index, 29
Recall, 28
relationship
  horizontal, 12
    precedence, 12
    sibling, 12
  vertical, 12
    ancestor-descendant, 12
    parent-child, 12
rule
  confidence, 26
  size, 26
  support, 26

S
Sensitivity, 28
set of profiles, 44
Silhouette coefficient, 30
Specificity, 28
subtree, 14, 78
  embedded, 14
  full subtree, 78
  induced, 14
Sum of squared errors, SSE, 29

T
training, 25
training examples, 25
tree, 13, 78
  labeled, 13, 78
  nodes
    child, 78
    empty node, 78
    leaf, 78
    parent, 78
    root, 13, 78
    siblings, 78
  ordered, 13, 78
  rooted, 13, 78
tree-edit distance, TED, 18
tree-edit operations
  deletion, 17, 78
  insertion, 17, 78
  relabeling, 17, 78
tree-edit sequence, 18
true negative, TN, 28
true positive, TP, 28

X
XCleaner2, 47
XML document, 12
  content, 2
  information types
    direct, 12
    explicit, 12
    implicit, 12
    indirect, 12
  structure, 2, 11
    attribute, 12
    element, 11, 12
XPattern, 41